University of Zurich[UZH]

Department of Informatics

Efficient Spectral Link Prediction on Graphs

Approximation of Graph Algorithms via Coarse-Graining

Dissertation submitted to the Faculty of Business,
Economics and Informatics
of the University of Zurich

to obtain the degree of
Doktor der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, PhD)

presented by
Daniel Spicar
from Zürich, ZH, Switzerland

approved in September 2019

at the request of
Prof. Dr. Abraham Bernstein
Prof. Dr. Claudio Tessone

August, 2021

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zurich, September 18, 2019

The Chairman of the Doctoral Board: Prof. Dr. Michael Böhlen

Abstract

Spectral graph theory studies the properties of graphs in relation to their eigenpairs, that is, the eigenvalues and eigenvectors of associated graph matrices. Successful applications of spectral graph theory include the ranking of web search results and link prediction in graphs.

Link prediction is the prediction of connecting edges between vertices of a graph or nodes of a network. It is used to predict the evolution of graphs and to discover previously unobserved edges. However, the computation of eigenpairs is computationally very demanding because the eigenvalue-eigenvector decomposition of graph matrices has cubic time complexity. As graphs or networks become large, this makes the computation of a full eigendecomposition infeasible. This work addresses the complexity problem for a state-of-the-art spectral link prediction method which is among the most accurate methods currently available but limited to relatively small graphs because several eigenvalue-eigenvector decompositions are required to compute a result.

Like previous work on similar complexity bottlenecks, this thesis approaches the problem by computing only a subset of the eigenpairs in order to obtain an approximation of the original method at lower computational cost. However, instead of modifying the original link-prediction algorithm, it uses the eigenpair subset to approximate the graph without significant changes to the link prediction algorithm. The graph is approximated by spectral coarse-graining, a method that shrinks graphs while preserving their dominant spectral properties. This approach is motivated by the hypothesis that results computed on a coarse-grained graph approximate the original link prediction results.

The main contribution presented in this dissertation is a new, coarse-grained spectral link-prediction approach. In a first part, the state-of-the-art link prediction method is combined with spectral coarse-graining. The computational cost, complexity, and link prediction accuracy is then evaluated. Theoretical analysis and experiments show that the coarse-grained approach produces accurate approximations of the original method with a significantly reduced time complexity. Thereafter, the spectral coarse-graining method is extended to make the complexity reduction more controllable and to avoid the computation of the eigenvalue-eigenvector decomposition. This dramatically increases the efficiency of the proposed approach and allows to compute more accurate graph approximations. As a result, the link prediction accuracy can be significantly improved while maintaining the reduced time complexity benefits of the coarse-grained approach.

Furthermore, the proposed approach produces a valid graph of the same structure and type as the original graph. In principle, it can be used with many other graph applications without the need for major adaptations. While a detailed evaluation is outside the scope of this thesis, the presented work is a step towards a more general approximation framework for spectral graph algorithms.

Zusammenfassung

Die spektrale Graphentheorie untersucht die Eigenschaften von Graphen in Bezug auf ihre Eigenpaare, d.h. die Eigenwerte und Eigenvektoren der zugehörigen Adjazenzmatrix. Erfolgreiche Anwendungen umfassen die Rangierung von Webseiten in Suchmaschinen und die Vorhersage von Kanten in Graphen.

Kantenvorhersage in Graphen ist die Vorhersage von Verbindungen zwischen Knotenpunkten von Graphen oder Netzwerken. Damit kann die Entwicklung von Graphen vorhergesagt werden oder es können unbeobachtete Kanten entdeckt werden. Die Berechnung von Eigenpaaren ist jedoch sehr komplex, denn die Zerlegung einer Matrize in alle ihre Eigenwerte und Eigenvektoren hat kubische Komplexität. Wenn Graphen oder Netzwerke sehr gross werden, ist die Berechnung einer vollständigen Eigenwertzersetzung nicht mehr möglich. Die vorliegende Arbeit löst dieses Komplexitätsproblem für ein Verfahren zur Kantenvorhersage welches zu den genauesten gehört. Allerdings erfordert es mehrere Eigenwert-Eigenvektor-Zerlegungen und ist daher nur auf relativ kleine Graphen anwendbar.

Verwandte Arbeiten zu ähnlichen Komplexitätsproblemen berechnen nur eine Teilmenge der Eigenpaare um dieses Problem zu lösen. Die Idee dahinter ist, dass dies zu einer Annäherung an die ursprüngliche Methode mit wesentlich kleinerem Rechenaufwand führt. Die vorliegende Arbeit verfolgt den gleichen Ansatz, aber sie belässt den Algorithmus fast unverändert. Statt dessen wird der Graph mithilfe einer spektralen Grobkörnung angenähert. Dies is ein Verfahren zum Verkleinern von Graphen wobei die dominanten spektralen Eigenschaften beibehalten werden. Diesem Ansatz die Hypothese zugrunde, dass auf einem grobkörnigen Graphen berechnete Ergebnisse die ursprünglichen Ergebnisse approximieren.

Der wichtigste Beitrag dieser Dissertation ist ein neuer, spektraler Algorithmus zur Kantenvorhersage. In einem ersten Teil wird ein etablierter Algorithmus mit spektraler Grobkörnung kombiniert und der Berechnungsaufwand, die Komplexität und die Genauigkeit der Vorhersage ausgewertet. Theoretische Analysen und Experimente zeigen auf, dass dieser Ansatz genaue Annäherungen an die ursprüngliche Methode mit einer deutlich reduzierten Zeitkomplexität ermöglicht. Danach wird die spektralen Grobkörnung erweitert, um die Komplexitätsreduktion besser kontrollieren zu können und um die Berechnung der Eigenwert-Eigenvektor-Zerlegung zu vermeiden. Dies erhöht die Effizienz des vorgeschlagenen Algorithmus drastisch und ermöglicht die Berechnung genauerer Graphenannäherungen. Dadurch kann die Genauigkeit der Kantenvorhersage deutlich verbessert werden, während die reduzierte Zeitkomplexität beibehalten wird.

Darüber hinaus erzeugt der vorgeschlagene Ansatz immer Graphen der gleichen Struktur und Art wie der ursprüngliche Graph. Im Prinzip ermöglicht dies den selben Ansatz mit vielen weiteren Graphenalgorithmen zu verwenden, ohne dass grössere Anpassungen erforderlich währen. Eine genaue Untersuchung weiterer Algorithmen sprengt den Rahmen dieser Thesis. Trotzdem kann der vorgestellte Ansatz als Schritt zu einem allgemeinen Methode für spektrale Graphenalgorithmen betrachtet werden.

Acknowledgements

First of all, I thank my advisor Abraham Bernstein for the opportunity to pursue doctoral studies in his research group at the University of Zurich. His support and patience made this thesis possible and provided me with precious knowledge and experiences.

I thank Claudio Tessone for his valuable advice that initiated an important part of the research presented in this thesis and for taking the time to co-referee my dissertation.

I thank Daniele Dell'Aglio for his mentorship, valueable input to this thesis, and for his friendship.

My studies at the Institute of Informatics have been a challenging experience. But thanks to my friends and colleagues who I will gladly remember this time. In particular I want to express a heartfelt thank you to Bibek, Christian, Cosmin, Cristina, Daniel, Ela, Helen, Juk, Katrin, Koa, Lorenz, Malena, Marc, Martin, Michael, Narges, Patrick de Boer, Patrick Minder, Pengcheng, Philip, Romana, Sebastian, Shen, Suzanne, Thomas, Timo, Wen, and Yasett.

Finally, without the love and support of my wife Věra I could not have finished this thesis. I will be forever grateful for her persistent support and sacrifice during the last months. I am fortunate to have found a great partner in her.

Table of Contents

Chapter 1

Introduction

1.1 Introduction

Graphs are a versatile and intuitive concept used in mathematics to model relations between objects. An informal definition of a graph is "a collection of interconnected things". More formally, a *graph* is a structure consisting of objects and relations. Objects are modeled as *vertices* and relations are represented by *edges*. When graphs manifest in a practical context they are often called *networks*. The vertices and edges of networks are also called *nodes* and *links* respectively.

The first formal use of graph theory is attributed to Leonhard Euler who developed its foundations in 1736 for a proof of the famous Seven Bridges of Königsberg problem. This mathematical puzzle asked for a path trough the city that crosses each of its (at the time) seven bridges exactly once. Euler had the idea to ignore the geometry implied by the location of the bridges and landmasses. Instead, he isolated only the relevant topological features in a graph model of Königsberg and presented an elegant proof that such a path does not exist.

Since Euler, graph models have been adopted in a wide range of problems. For example, in chemistry the energetic states of subatomic particles can be modeled as graphs and in computer science, the ranking of web search results is framed as a graph problem. One advantage of graphs is that they are simple to adapt to different contexts because only a definition of objects and relationships is required to define a graph model. When relationships are not explicit, they can almost always be constructed, for example, by the definition of an order or topology that gives rise to neighborhood relations and distances.

However, the apparent simplicity of graph models is deceptive. The interaction space of a large number of objects is high-dimensional. The graphs used in this thesis define relations as numerical values that can be represented by a matrix. Suppose there are N vertices in a graph. To represent all relations between each pair of vertices, each graph vertex is associated to exactly one row and one column which gives rise to a matrix with dimensions $N \times N$. This matrix represents an interaction space containing N^2 possible edges, that is, every vertex can have a relation to every other vertex and itself. Algorithms that explore this interaction space are generally of exponential complexity and become infeasible to compute as graphs grow large.

Computational complexity is a particular concern with applications that rely on spectral graph theory, a field of mathematics that associates the eigenvalues and eigenvectors of graph matrices to the structure of graphs. The use of spectral properties of graphs originates from chemistry where atomic bonds can be represented as graphs and their eigenvalues relate to energies contained in these systems. In physics and mathematics, spectral graph theory is used to obtain discrete solutions to partial differential equations,

for example, when modeling the forces that act on atoms in vibrating membranes. More recently, spectral graph theory has been adopted in computer science where it plays an important role in data clustering (Cheeger, 1969), searching and ranking (Page et al., 1999), and data privacy in social networks (Ying and Wu, 2008). Cvetković and Simić (2011) published an extensive survey of spectral graph theory applications in computer science.

The main reason for the high computational complexity of many spectral graph theory applications is that the eigendecomposition of a graph matrix, that is, the computation of all of its eigenvalues and eigenvectors, has complexity $O(N^3)$ (see Section 2.4). This means that it becomes infeasible to compute all the spectral properties of large graphs. The definition of "large" in this context is changing constantly as computers become more capable. Furthermore, applications differ in whether they require a subset or all of the spectral properties. Generally, graphs with thousands or tens of thousands of vertices pose a practical limit for the computation of a full eigendecomposition on commodity hardware (Demmel, 1997, Saad, 2011). In high performance computing environments significantly larger graphs can be processed but the the complexity problem remains the same. This dissertation focuses on the former case.

Industry experts and researchers have repeatedly overcome these computational limits with clever algorithm design and exploitation of problem-specific circumstances (e.g., Cvetković et al., 1999, Page et al., 1999), or with approximations whose accuracy is comparable to or indistinguishable from exact solutions (Bai et al., 2000, Demmel, 1997, Trefethen, 2013). However, these approaches are difficult to develop and solutions may not be transferable to other applications.

In this dissertation the computational complexity problem is addressed by reducing graph complexity with spectral coarse-graining (SCG, de Lachapelle et al., 2008). SCG is a method that shrinks graph matrices while preserving a subset of their spectral properties. The intention of this approach is to work towards an approximation framework that is more generally applicable than application-specific designs. Given any application that depends on spectral properties of an input graph, ideally, the application result can be approximated at lower computational complexity without changes to its algorithm. The approximation occurs because the input graph is replaced with a smaller coarse-grained graph that preserves the most important spectral properties. In principle, any algorithm is suitable as the coarse-graining primarily changes the graph size but it preserves the graph semantics and dominant structure.

While the approach is general in the sense that it can be applied to any problem that takes a suitably structured graph as input; not every applications is amenable to approximation by coarse-graining. The shrinking of the input graph necessarily incurs a loss of information. Depending on the application, the removal of fine-grained information is more or less harmful. Spectral coarse-graining can be interpreted as dimensionality reduction for graphs (de Lachapelle et al., 2008). Similar techniques are widely used in machine learning tasks and statistical contexts (Abdi and Williams, 2010), therefore, a coarse-graining approach is expected to work well on applications that use the graph structure as a feature in a machine learning pipeline or in statistical applications.

In this dissertation, the scope is limited to a link prediction method that serves as a use-case for the proposed framework. Given an observation of a graph, link prediction attempts to predict the existence of unobserved edges. This process can be interpreted as a forecast of the evolution of a graph, or equivalently, as the discovery of edges that are missing due to incomplete information. Link prediction is used in recommender systems which have become widely used in industry. Examples include friendship recommendations in online social networks or product suggestions in web stores. The other view of link prediction as the discovery of unobserved edges has been proposed for situations where observations are challenging, uncertain, or costly. They occur in natural sciences, for example, when studying metabolic systems. In metabolic networks, edges represent complex biochemical pathways. New observations require resource-intensive experimentation in laboratories. Link prediction can identify holes in metabolic pathways and propose promising candidate edges. As a consequence, resources can be used more effectively by directing the attention of scientists to promising experiments. A detailed introduction to link prediction is given in Section 3.1.

The structural perturbation method for link prediction (SPM, Lü et al., 2015) is chosen as a use-case for the coarse-grained approach proposed in this thesis. SPM is one of the most accurate spectral link-prediction methods (Pech et al., 2017, Zeng et al., 2018c) and an ideal use-case because it depends directly on the eigenvalues and eigenvectors of graph matrices. Furthermore, it showcases the problem of high complexity of spectral graph algorithms. Several studies have found that SPM is severely limited by its large computational complexity (Muscoloni and Cannistraci, 2017, Pech et al., 2017). The SPM algorithm remains mostly unmodified in order to uphold the claim of general applicability of the proposed approach. However, a comprehensive study of its generalizability is beyond the scope of this dissertation and deferred to future work.

The problem addressed by this dissertation can be summarized as the design of a computationally efficient framework of methods for the approximation of an input graph while preserving its most important spectral properties. The proposed solution is suitable for applications that depend of the eigendecomposition of the input graph. This dissertation chooses a link prediction use-case to demonstrate how the proposed framework can be used to reduce algorithm complexity and computational cost. This yields a new spectral link prediction method that can approximate SPM link prediction at a significantly lower computational cost.

1.2 Research Questions and Hypotheses

In order to verify some of the hypotheses below, the presented contributions are evaluated on *realistic* graphs. That means graphs used for link prediction in realistic applications or data collected from real observations and experiments. Sometimes, such graphs are called "real world" graphs to distinguish them from synthetic graphs generated from (random) graph models.

This thesis addresses the following research questions and hypotheses.

Research Question 1 *Is SPM link prediction on coarse-grained graphs a viable approach to significantly reduce the high computational cost of SPM link prediction?*

The viability of an approach can be defined in various ways. In this dissertation, it is interpreted as a beneficial trade-off between link-prediction accuracy and computational cost reduction. This condition is specified by the following hypotheses associated to Research Question 1.

Hypothesis 1.1 *SPM link prediction on coarse-grained graphs can exploit beneficial trade-offs between link prediction accuracy and computational cost.*

A beneficial trade-off incurs a lower loss of accuracy than the reduction of computational cost it achieves. In order to evaluate this hypothesis, the computational cost of SPM is related to the graph size. Then, link prediction accuracy is evaluated as a function of graph size to show the existence of beneficial trade-offs in regions where this function has a small slope.

Hypothesis 1.2 *SPM link prediction on coarse-grained graphs reduces the time complexity of SPM link prediction.*

To verify this hypothesis, the computational cost of the SPM implementation is quantified by an analysis of its floating point operation count. The same is done for the coarse-grained SPM implementations. From these analyses a time complexity can be estimated.

Research Question 2 *Can spectral coarse-graining (SCG) be defined without knowledge of the eigenvalues and eigenvectors of a graph?*

Since one of the limiting factors of SPM link prediction and SCG is the high computational cost of the eigendecomposition, it is very desirable to circumvent it entirely. SCG, as defined by de Lachapelle et al. (2008), preserves the spectral features of a graph but requires that the subset of eigenvectors that is preserved is known. Therefore, only few eigenvectors can be preserved because their computation is costly. The intention of Research Question 2 is to explore the possibility to define SCG using a less expensive proxy for the eigenvectors.

Hypothesis 2.1 *Polynomial expansion filtering and random projections can be used to compute a spectral vertex similarity index suitable for spectral coarse-graining without requiring an eigendecomposition and with lower computational cost.*

Spectral coarse-graining uses eigenvectors to embed vertices in a spectral feature space. Vertices that are located close to each other in this feature space are grouped together and represented by a single super-vertex. Any feature space that preserves pairwise distances between the vertex embeddings can be used to yield an equivalent grouping. Different applications have used a technique called polynomial expansion filtering to obtain a distance-preserving proxy for the spectral features of matrices, for example, for principal component analysis (Ramasamy and Madhow, 2015) or spectral clustering (Tremblay et al., 2016b). Hypothesis 2.1 can be investigated using mathematical theory and with detailed analysis of its computational cost.

Hypothesis 2.2 *Given that the coarse-grained graph size is equal, the preservation of a large number of eigenspaces in a spectral coarse-graining improves link prediction accuracy on coarse-grained graphs.*

Since the circumvention of the eigendecomposition is expected to reduce the computational cost, some of the resulting efficiency gain can be used to improve link prediction accuracy. Hypothesis 2.2 states that this can be done by the preservation of a larger number of spectral features in the coarse-graining. This hypothesis can be evaluated experimentally with a comparison of the link-prediction accuracy and runtime of the proposed approach to a coarse-grained SPM with standard SCG.

1.3 Summary of Contributions

This section summarizes the most relevant contributions proposed in this thesis.

SPM Eigenvector Stability Analysis: SPM link prediction relies on an assumption of invariant eigenvectors under small perturbations. However, the conditions for this assumption to hold are not investigated or verified in related work. In Section 3.2.2, these conditions are established by connecting them to error bounds known from matrix perturbation theory. This makes some conditions for the validity of the assumption explicit and shows when the assumption can break down.

SPM Complexity Analysis: The computational cost and bottlenecks of SPM have not been analyzed in detail in related work. In Section 3.4.2, the operation count for SPM link prediction on dense and sparse graphs as well as for degenerate and non-degenerate spectra is established. Then, in Section 3.4.3, the time complexity of SPM is determined. An in-depth analysis allows to identify the computational bottlenecks of the approach precisely.

SCG Error Bounds for Real Matrices: New error bounds for the approximation error induced by spectral coarse-graining are derived in Section 4.3.5. These bounds become important for the assessment of coarse-grained SPM link prediction runtimes and for the interpretation of accuracy results.

Coarse-Grained SPM (CGSPM): The coarse-grained SPM link prediction approach is defined and evaluated in Chapter 5. The proposed CGSPM approach is a new method for the approximation of SPM link prediction that allows an application to trade-off prediction accuracy for lower computation cost. Furthermore, its operation count and complexity is analyzed in detail and an extensive parameter space search is conducted to determine strategies and heuristics for parameters that yield good results. Further experiments verify that significant reductions in algorithm runtime are possible while maintaining high prediction accuracy.

Efficient and Controllable SPM (ECSPM): A new, more efficient extension to spectral coarse-graining is developed in Chapter 6. This extension circumvents the requirement to compute an eigendecomposition and enables ECSPM to preserve a larger number of spectral properties in the coarse-graining. Furthermore, ECSPM makes the coarse-grained graph size a method parameter which addresses a problem of CGSPM where this size is not directly controllable. The operation count and time complexity of ECSPM is analyzed in detail and its link prediction runtime and accuracy is evaluated in a series of experiments. ECSPM is more accurate and at least as efficient as CGSPM.

1.4 Outline

This thesis is organized in two parts.

The first part contains preliminaries. It introduces basic definitions, fundamental theories, and introduces two methods upon which the remainder of this dissertation relies. In Chapter 2, selected topics of graph theory and spectral graph theory are defined and the complexity of the spectral decomposition is discussed. Additionally, eigenpair perturbation theory, Chebyshev polynomial approximation theory, and classifier evaluation in ROC space are defined and explained. In Chapter 3, the structural perturbation method for link prediction (SPM) is explained and analyzed in detail. This link prediction method is the use-case and application for contributions proposed in this dissertation. In Chapter 4, spectral coarse-graining (SCG) is defined and many of its properties are evaluated and discussed. SCG is the framework used to compute graph approximations and forms the basis of the main contributions presented in this thesis.

The second part presents the main contributions developed in the scope of this dissertation. In Chapter 5, a new method called coarse-grained SPM (CGSPM) is proposed to address Research Question 1. This approach is a combination of SPM and SCG and allows to approximate SPM at a significantly lower computational cost. This contribution fully confirms Hypothesis 1.1 and partially confirms Hypothesis 1.2. In Chapter 6, an extension to spectral coarse graining and CGSPM is presented. The efficient and controllable SPM (ECSPM) method addresses drawbacks of CGSPM and increases the efficiency of SCG with a polynomial approximation approach that allows to circumvent one of the bottlenecks of CGSPM. This contribution addresses Research Questions 1 and 2 and confirms all the hypotheses formulated in in Section 1.2.

Finally, Chapter 7 concludes this thesis with a review of the research questions and a discussion of future works.

Part I

Preliminaries

Chapter 2

Definitions and Background

This chapter defines terms, notation, and elements of the fundamental theory of graphs and their spectra. Thereafter, selected topics are introduced and summarized to provide context and definitions related to specific aspect of the work presented in the following chapters.

Section 2.1 states basic definitions of matrix theory and the notation conventions used in this thesis. Basic graph theory is defined in Section 2.2 and extended to spectral graph theory in Section 2.3. Section 2.4 discusses the complexity of the eigenvalue problem. The content of these sections is fundamental and pertains to all parts of this dissertation.

Section 2.5 contains a short introduction to matrix perturbation theory, upon which the contents of Chapters 3 and 5 rely. Subsection 2.5.1 contains a mathematical derivation of the approximation of eigenpair perturbation terms. It can be considered optional reading for readers wishing to fully understand the link prediction algorithm presented in Chapter 3. Section 2.6 is a self-contained summary of the polynomial approximation theory which is used to state the contributions in Chapter 6. Finally, Section 2.7 presents a self-contained summary of binary classifier evaluation with receiver operating characteristics (ROC). While ROC curves are used commonly in classifier evaluation, this section defines how ROC curves are averaged and how measures of ROC curve variance are calculated in the context of this dissertation.

2.1 Definitions and Notation

Indices are generally defined by sequences $i = 1, \ldots, n$ or sets $j = \{0, \ldots, n\}$, where n is a non-negative integer. Without explicit specification the enumeration increment is assumed to be 1, that is, the sequence for i defined before includes all strictly positive integers from 1 to n.

The *Kronecker delta* is used in the definition of different mathematical relationships. It indicates whether the two variables are equal. The arguments are written in index notation due to its frequent use in the comparison of indices,

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

The set of all real numbers is denoted by $\mathbb{R}$. Furthermore, the space of all column vectors with n real components is $\mathbb{R}^n$ and the set of all matrices with m rows and n columns with real elements is $\mathbb{R}^{m \times n}$.

Scalars and Vectors: Variables are denoted with single letters (e.g., c). Depending on context, they denote either scalars or vectors. Without further qualification, all vectors are considered column vectors. That means $x \in \mathbb{R}^k$ is a vector equivalently defined as

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix}$$

and $x = (x_1, \ldots, x_k)^\top$, where $^\top$ denotes the transpose. Vectors are normally indexed with a subscript letter such as $y_i \in \mathbb{R}^n$. Vector components can be denoted in parentheses such that $y_i(j)$ is the j-th component of y_i.

Matrices: Matrix elements are typeset in capital letters such that M_{ij} is the element located at the i-th row and the j-th column. Matrices are always denoted in boldface capital letters such that $\mathbf{M} \in \mathbb{R}^{m \times n}$ is a real matrix of the form

$$\mathbf{M} = \begin{pmatrix} M_{11} & M_{12} & \cdots & M_{1n} \\ M_{21} & M_{22} & \cdots & M_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ M_{m1} & M_{m2} & \cdots & M_{mn} \end{pmatrix}.$$

Matrices can be defined as a concatenation of column vectors. The definition $\mathbf{R} = [\ r_1\ r_2\ \cdots\ r_n\ ]$ with column vectors $r_i \in \mathbb{R}^m$ and $i \in \{1, \ldots, n\}$ yields a matrix of the same shape as $\mathbf{M}$. The i-th column of a matrix is indexed as $\mathbf{M}(i)$. The transpose of a matrix is denoted $\mathbf{M}^\top$. Powers of a matrix are written $\mathbf{M}^n = \underbrace{\mathbf{M} \cdots \mathbf{M}}_{n}$.

The rank of a matrix is denoted $\text{rank}(\mathbf{M})$ and defined as the dimensionality of its column space,

$$\{\mathbf{M}x : x \in \mathbb{R}^n\},$$

which is the set of all linear combinations of the columns of $\mathbf{M}$. The column space of $\mathbf{M}$ is equivalently defined as the subspace spanned by the columns of $\mathbf{M}$, or $\text{span}(\mathbf{M})$.

Special Matrices: Diagonal matrices are written $\mathbf{D} = \text{diag}(x_1, x_2, \ldots, x_k)$ which defines a square matrix $\mathbf{D} \in \mathbb{R}^{k \times k}$ with the diagonal elements set to the vector $(x_1, x_2, \ldots, x_k)$ and all off-diagonal elements 0. A special diagonal matrix is the identity matrix written $\mathbf{I}_n$ which defines a square $n \times n$ matrix with all diagonal elements set to 1 and all other elements set to 0. Another special matrix is the zero-matrix $\mathbf{0}$ (boldface zero) whose dimensions can be derived from the context. $\mathbf{M}^{-1}$ denotes the *matrix inverse* defined by $\mathbf{M}\mathbf{M}^{-1} = \mathbf{I}$.

Norms: The following norms are synonymous: 2-norm, Euclidean norm, least-squares norm, vector length, or $\mathcal{L}_2$ norm. This norm is denoted $\|x\|$, where x is a vector in $\mathbb{R}$ and defined as

$$\|x\| = \sqrt{x^\top x} = \sqrt{x_1^2 + x_2^2 + \cdots + x_k^2}.$$

The matrix norm $\|\mathbf{M}\|$ for a $m \times n$ matrix is called the *spectral norm* and defined,

$$\|\mathbf{M}\| = \sigma_{\max},$$

where $\sigma_{\max}$ is the largest singular value. For any symmetric matrix $\mathbf{A}$, the largest singular value is equal to the largest eigenvalue, $\|\mathbf{A}\| = \lambda_{\max}$ (see Section 2.3).

Some useful properties and definitions used throughout this thesis are listed below.

- A *unit vector* has length one: $a^\top a = \|a\| = 1$.
- Two vectors are *orthogonal*, when their inner product is zero: $a^\top b = \|ab\| = 0$.
- A *square matrix* has the same number of rows and columns, i.e. $m = n$.
- $\mathbf{A}$ is *symmetric* when it is equal to its transpose: $\mathbf{A} = \mathbf{A}^\top$. This implies $\mathbf{A}$ is square.
- $\mathbf{U}$ is *orthogonal*, when its columns and rows are orthogonal unit vectors. Then $\mathbf{U}^\top\mathbf{U} = \mathbf{U}\mathbf{U}^\top = \mathbf{I}$ and $\mathbf{U}^\top = \mathbf{U}^{-1}$.

Further definitions are introduced in the main text as needed in a particular context.

2.2 Basic Graph Theory

Graphs are mathematical structures modeling the interactions between pairs of objects. Each modeled object is represented by a *vertex*. In principle, any vertex can have an interaction with any vertex, including itself. When a vertex interaction is modeled, it is represented by an *edge* connecting exactly two vertices. A graphical representation of a graph with five vertices and six edges is shown in Figure 2.1a.

Suppose $G(V, E)$ is a graph with *vertex set* $V = \{v_1, \ldots, v_n\}$ and *edge set* E such that the pair $(v_i, v_j) \in E$ if vertex v_i is connected to vertex v_j. The size of a graph is the number of vertices. Throughout this work, the graph size is typically denoted by the variable N defined as $N = |V|$. All graphs in this work are assumed to be finite, undirected, and without multiple edges. In undirected graphs any edge (v_i, v_j) is an unordered pair. The following edges are equal,

$$(v_i, v_j) = (v_j, v_i) \qquad \forall i, j \in \{1, \ldots, N\}.$$

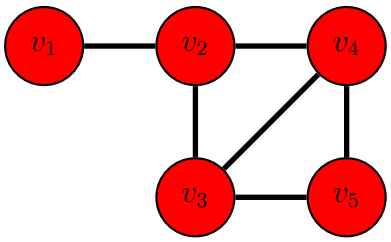
Furthermore, G is strongly connected. That means any vertex is reachable from any other vertex by a series of edge traversals. Some graphs have *loops*, i.e. edges of the form (v_i, v_i) that connect a vertex to itself. A graph G is assumed to have no loops. Otherwise it is discussed in context.

Any graph $G(V, E)$ can be fully represented by a square *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{N \times N}$ that represents the interactions between directly connected vertices. The elements of $\mathbf{A}$ are defined by

$$A_{ij} = \begin{cases} w(i,j) & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise,} \end{cases}$$

with $w : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a function that assigns a *weight* to the interaction of v_i and v_j. Throughout this dissertation

$$w(i, j) = 1 \qquad \forall i, j \in 1, \ldots, N$$



(a) *A small graph.*

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

(b) *The adjacency matrix of (a).*

Fig. 2.1

such that the adjacency matrix contains only values 0 and 1. Exceptions are discussed in context. Because all edges are undirected, $\mathbf{A}$ is necessarily symmetric and $A_{ij} = A_{ji}$. Without loops, the diagonal of $\mathbf{A}$ contains only zeros.

By definition of the adjacency matrix the rows and columns of $\mathbf{A}$ and the vertices $v \in V$ are associated by their index. Row i of $\mathbf{A}$ is a vector $a = (A_{11}, A_{12}, \ldots, A_{1N})$ describing the interactions of vertex v_i with all vertices (including itself). The same association exists for column $\mathbf{A}(i)$ and vertex v_i. Figure 2.1b shows an example of an adjacency matrix.

2.3 Spectral Graph Theory

Spectral graph theory studies the properties of graphs in relation to the eigenvalues and eigenvectors of their matrix representation. This section presents selected topics of spectral graph theory which are referenced throughout this dissertation.

2.3.1 Eigenvalues and Eigenvectors of Graphs

Let matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ be symmetric and contain only non-negative elements. Any scalar $\lambda \in \mathbb{R}$ is an *eigenvalue* of $\mathbf{A}$ if it satisfies the *eigenvalue equation*

$$\mathbf{A}u = \lambda u \tag{1}$$

for some vector $u \in \mathbb{R}^N$. Each non-zero vector u satisfying (1) is called an *eigenvector* of $\mathbf{A}$ corresponding to eigenvalue λ. Every solution to the eigenvalue equation is an *eigenpair* (λ, u). The set of all eigenvalues is called the *spectrum* of $\mathbf{A}$ and the eigenvalue equation is also known under different names such as *eigenvalue-eigenvector equation*, *eigenvalue problem*, or *eigenproblem*.

Because the adjacency matrix is a complete representation of graph G, the reader will sometimes encounter expressions that use the adjacency matrix and the graph synonymously. For example, a statement can reference the eigenpairs of graph G by which the eigenpairs of the adjacency matrix of G are meant.

Theorem 2.1 (Spectral Theorem for Real Symmetric Matrices). *Let $\mathbf{A} \in \mathbb{R}^{N \times N}$ and $\mathbf{A} = \mathbf{A}^\top$. Then there exists an orthogonal matrix $\mathbf{U} \in \mathbb{R}^{N \times N}$ and a diagonal matrix $\mathbf{\Lambda} \in \mathbb{R}^{N \times N}$ such that*

$$\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top. \tag{2}$$

A proof can be found in Horn and Johnson (1985, Section 4.1.5).

As a corollary of Theorem 2.1, there is a basis consisting of N mutually orthogonal and real unit eigenvectors and corresponding real eigenvalues for any real symmetric matrix $\mathbf{A}$. Equation (2) is called the *eigendecomposition*, *spectral decomposition*, or more generally, a *diagonalization* of $\mathbf{A}$.

Throughout this work the eigenvalues are ordered from largest to smallest such that

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N.$$

The matrix $\mathbf{U}$ is referred to as the *eigenvector matrix* or *eigenbasis* and it is defined by the concatenation of mutually orthogonal eigenvectors in order of the corresponding eigenvalues,

$$\mathbf{U} = [\ u_1\ u_2\ \cdots\ u_N\ ].$$

If an eigenvalue is repeated, i.e. there exist two eigenvalues such that $\lambda_i = \lambda_j$ with $i \neq j$, then the *multiplicity* of that eigenvalue is larger than one and the spectrum is said to be *degenerate*. Otherwise, the eigenvalue is called *simple* and the spectrum is *non-degenerate*. A *zero eigenpair* is an eigenpair with $\lambda = 0$. Throughout this work references to the *leading* eigenvectors or eigenpairs are used to denote those with largest corresponding eigenvalue.

Finally, Equation (2) can be rewritten as a sum over all eigenpairs

$$\mathbf{A} = \sum_{i=1}^{N} \lambda_i u_i u_i^\top. \tag{3}$$

2.3.2 Vector Spaces Spanned by Eigenvectors

The N eigenvalues of $\mathbf{A}$ are not necessarily all distinct. The span of eigenvectors corresponding to the same eigenvalue λ is called the *eigenspace* of λ. When λ is simple, the eigenspace corresponds to a line in $\mathbb{R}^N$ and its eigenvector is uniquely defined as a unit vector along this line. Suppose λ has multiplicity $m > 1$, meaning that m eigenvectors correspond to an eigenvalue with the same arithmetic value. Then, the eigenspace of λ has m dimensions and infinite possibilities exist to choose unit eigenvectors that are not necessarily orthogonal in this eigenspace. One of the important corollaries of Theorem 2.1 is that N mutually orthogonal eigenvectors can be chosen and the columns of $\mathbf{U}$ are defined as mutually orthogonal eigenvectors, even when they correspond to the same eigenvalue.

The column space of the eigenvector matrix $\mathbf{U}$ is called the *eigenbasis* associated to $\mathbf{A}$ and it is an orthonormal basis in $\mathbb{R}^N$ by construction. Furthermore, $\mathbf{U}$ is an orthogonal matrix. However, much of this thesis considers the span of eigenvectors corresponding to the eigenspaces of a subset of eigenvalues. These eigenvalue subsets induce linear subspaces of the eigenbasis that are themselves orthonormal bases in some subspace of $\mathbb{R}^N$. The terminology tends to become convoluted in these situations.

To facilitate discussions, the term *eigenspace* is sometimes used imprecisely in this dissertation. The reader will encounter expressions such as "the eigenspace of $\mathbf{A}$" which refer to the column space of $\mathbf{U}$, that means, sometimes the word eigenspace (singular) is

used to refer to the subspace spanned by all eigenvectors. In such contexts a subspace of an eigenspace, or *eigensubspace*, refers to the span of a subset of eigenvectors.

The eigenvector matrices or sets of eigenvectors can be called eigenspaces although they are not proper vector spaces because the zero vector is not an eigenvector. In these contexts an inner product space is implicitly assumed.

2.3.3 Orthogonal Projections onto Eigenspaces

Many of the topics discussed in this thesis are concerned with othogonal projections, in particular projections onto eigensubspaces. The following discussion defines these projections and makes explicit that the orthogonality of $\mathbf{U}$ implies that projections onto eigenspaces are orthogonal projections that minimize the vector 2-norm of the error.

Suppose a linear subspace of $\mathbb{R}^N$ is spanned by the $k \leq N$ largest eigenvectors of $\mathbf{A}$. The corresponding eigenvector matrix is $\mathbf{U}_k \in \mathbb{R}^{N \times k}$ and $\mathbf{U}_k = [\, u_1 \; u_2 \; \cdots \; u_k \,]$. By construction, the columns of $\mathbf{U}_k$ define a k-dimensional subspace of $\mathbb{R}^N$ and form an orthonormal basis in $\mathbb{R}^N$ corresponding to this subspace.

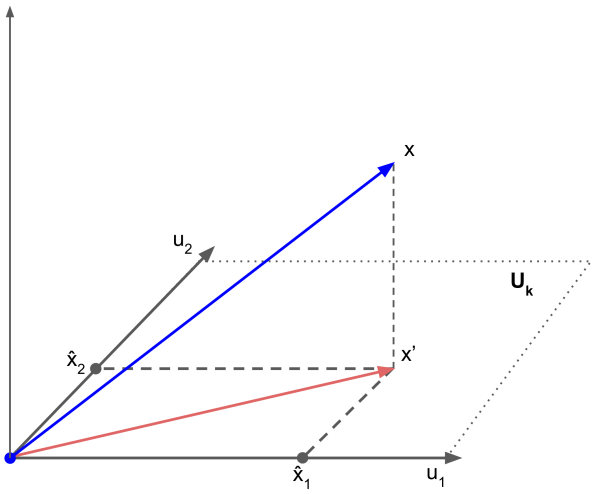


Fig. 2.2: *A sketch of an orthogonal projection of a vector $x \in \mathbb{R}^3$ onto a subspace spanned by two orthogonal vectors u_1 and u_2.*

Suppose a vector $x \in \mathbb{R}^N$ is projected onto the subspace spanned by the columns of $\mathbf{U}_k$. At this point the projection needs to be formalized mathematically. Consider the situation in Figure 2.2. A vector $x \in \mathbb{R}^3$ (blue) is projected onto a subspace spanned by the eigenvectors u_1 and u_2. Because the eigenvectors are linearly independent, the subspace is an infinite plane in $\mathbb{R}^3$. The projection of x is the vector x' (red). Notice that the difference $x - x'$ is another vector orthogonal (perpendicular) to $\mathbf{U}_k$ because the "missing" dimension

is necessarily orthogonal to the columns of $\mathbf{U}_k$ by construction of the subspace and also because otherwise $x - x'$ can be reduced further as the shortest distance is the orthogonal vector.

Therefore, a projection can be formalized in arbitrary dimensions as follows. The *orthogonal projection* of x onto the columns space of $\mathbf{U}_k$ is a vector x' such that the difference $x - x'$ is orthogonal to $\mathbf{U}_k$. Writing this as an inner product, the condition becomes

$$\begin{aligned} \mathbf{U}_k^\top (x - x') &= 0. \\ \mathbf{U}_k^\top x - \mathbf{U}_k^\top x' &= 0 \\ \mathbf{U}_k^\top x &= \mathbf{U}_k^\top x'. \end{aligned} \tag{4}$$

By definition of the problem, vector x' is a linear combination of the chosen eigenvectors,

$$x' = \sum_{i=1}^{k} \hat{x}(i) u_i = \mathbf{U}_k \hat{x}, \tag{5}$$

for some vector $\hat{x} = (\hat{x}(1), \ldots, \hat{x}(k))^\top$. Substituting (5) into (4) and solving for $\hat{x}$,

$$\begin{aligned} \mathbf{U}_k^\top x &= \mathbf{U}_k^\top \mathbf{U}_k \hat{x} \\ (\mathbf{U}_k^\top \mathbf{U}_k)^{-1} \mathbf{U}_k^\top x &= \hat{x}. \end{aligned}$$

Now the orthonormality of the eigenvectors can be used, that is $\mathbf{U}_k^\top \mathbf{U}_k = \mathbf{I}_k$. Therefore,

$$\mathbf{U}_k^\top x = \hat{x}. \tag{6}$$

Notice that each component $\hat{x}(i)$ is the inner product $u_i^\top x$ for $i = \{1, \ldots, k\}$. By definition of the inner product for euclidean spaces, each component $\hat{x}(i)$ is the length of x' along the eigenvector u_i. This can be interpreted as the coordinates of x' in the subspace spanned by $\mathbf{U}_k$ (see. Figure 2.2).

Combining (5) and (6) yields the orthogonal projection,

$$x' = \mathbf{U}_k \mathbf{U}_k^\top x. \tag{7}$$

The orthogonal projection defines an *eigenprojector* (also called *spectral projector*),

$$\mathbf{P}_k = \mathbf{U}_k \mathbf{U}_k^\top, \tag{8}$$

with $\mathbf{P}_k \in \mathbb{R}^{N \times N}$. Due to the orthogonality of the columns of $\mathbf{U}_k$ this is an orthogonal projector and $\mathbf{P}_k = \mathbf{P}_k^\top = \mathbf{P}_k^2$. The eigenprojector is necessarily a rank k projector because $\mathbf{U}_k$ is constructed from k orthonormal eigenvectors. Therefore, $\text{rank}(\mathbf{P}_k) = \text{rank}(\mathbf{U}_k \mathbf{U}_k^\top) = \text{rank}(\mathbf{U}_k) = k$.

Two different projections of $x \in \mathbb{R}^N$ onto the column space of $\mathbf{U}_k$ are defined by the equations above.

Projection onto Orthonormal Basis: Equation (6) defines a projection of x onto the column space of $\mathbf{U}_k$ in *eigenspace coordinates*. It can be interpreted as an embedding of x in the eigenspaces of $\mathbf{A}$ spanned by the k leading eigenvectors. The result of this projection is a vector $\hat{x} \in \mathbb{R}^k$ such that $\hat{x} = \mathbf{U}_k^\top x$. Equivalently, the projection of a row vector $x^\top$ onto the subspace spanned by $\mathbf{U}_k$ is $\hat{x}^\top = x^\top \mathbf{U}_k$.

Ortgogonal Projection onto Subspace: Equation (7) defines the orthogonal projection of x onto the column space of $\mathbf{U}_k$ in *the same coordinate system* (typically the standard basis of Cartesian coordinates). This can be interpreted as the reverse projection of $\hat{x}$ onto the original vector space. The result of this projection is a vector $x' \in \mathbb{R}^N$ such that $x' = \mathbf{P}_k x$. Equivalently, the reverse projection of a row vector is $(x')^\top = \hat{x}^\top \mathbf{U}_k^\top$ and the orthogonal projection of a row vector is $(x')^\top = x^\top \mathbf{P}_k$.

The projection of multiple vectors with the same projector can be formulated as a matrix equation $\mathbf{X}' = \mathbf{P}_k \mathbf{X}$, where $\mathbf{X} \in \mathbb{R}^{N \times d}$ and $d \in \mathbb{Z}^+$.

It is worth reiterating that the orthogonal projection minimizes the error of the vector $x - x'$ in respect to the 2-norm. The projection is defined such that the projection of $x - x'$ onto $\mathbf{U}_k$ is a vector with $\|\mathbf{U}_k^\top (x - x')\| = 0$, that is, one that is orthogonal to $\mathbf{U}_k$.

2.3.4 Association between Eigenvector Components and Graph Vertices

For a graph $G(V, E)$, Section 2.2 defines the association of each vertex $v_i \in V$ to the i-th row of the graph adjacency matrix $\mathbf{A}$. This section extends this relationship to the eigenvectors. In particular, it is shown that any vertex v_i is projected onto the eigenspace of any λ_k at the coordinate $\lambda_k u_k(i)$. As λ_k is constant for each vertex projected onto the same eigenspace, it can be ignored without affecting relative distances.

By definition of the adjacency matrix, each element A_{ij} of $\mathbf{A}$ represents the interaction of vertex v_i with vertex v_j. This relationship is transferred to the eigenspaces of $\mathbf{A}$ by a projection of each row of $\mathbf{A}$ onto its eigenspaces.

Let row i of $\mathbf{A}$ be $a_i = (A_{i1}, \ldots, A_{iN})$. Its projection onto the eigenspace of any λ_k is a scalar,

$$a_i u_k = \left[\sum_{t=1}^{N} \lambda_t u_t(i) u_t^\top \right] u_k = \lambda_k u_k(i), \tag{9}$$

for all $i = 1, \ldots, N$ and $k = 1, \ldots, N$. The first equality is obtained from the eigendecomposition in Equation (3). The sum disappears due to the orthogonality of the eigenvectors.

The full embedding of a vertex v_i in the eigenspace, is given by the projection of row a_i onto all eigenspaces of $\mathbf{A}$.

$$a_i \mathbf{U} = \left(\lambda_1 u_1(i), \lambda_2 u_2(i), \ldots, \lambda_N u_N(i) \right). \tag{10}$$

The eigenvalues can be ignored if the projection needs to preserve only relative distances because for any given eigenspace u_k, the eigenvalue is constant for all i. Furthermore, this relationship does not hold for eigenspaces associated to a zero eigenvalue.

2.4 The Complexity of the Eigendecomposition

This dissertation presents different algorithms and methods that rely on either a full eigendecomposition or a partial eigendecomposition. The former is used to obtain all N eigenvalues and an associated orthonormal basis of eigenvectors of a real symmetric $N \times N$ matrix. The latter is used when only a subset of k eigenpairs is required. In this thesis, these are typically the eigenvalues with largest arithmetic value and their associated mutually orthogonal eigenvectors. The state-of-the-art is summarized in this section. This discussion is based on the work of Demmel (1997) which discusses all algorithms referenced below in detail.

In general, the full eigendecomposition is computed with so-called *direct* methods that store the full matrix in computer memory. They do not use sparse matrix storage schemes and cannot profit fully from matrix sparsity. The current standard for computing all eigenvalues and eigenvectors of relatively small matrices is the *QR Algorithm*. A faster algorithm that is used on larger matrices is referred to as *divide-and-conquer method*. However, all these methods require an initial tridiagonalization of the matrix which is of complexity $O(N^3)$. Under special circumstances, these algorithms can performs better but according to Pan et al. (1998) the complexity can be no better than $O(N^2 \log\ N)$.

Large eigenproblems are solved with *iterative* methods. These algorithms start with few initial vectors and approximate eigenvalues and eigenvectors from low-dimensional subspaces. Each iteration improves the precision of the approximation and the algorithms stop when they converge on a desired subset of eigenvalues and eigenvectors. The iterative approach allows to keep the memory requirements limited and to avoid storing the full matrix in memory. These strategies also enable the exploitation of matrix sparsity. Extreme eigenvalues, i.e. those located at either end of the spectrum and well-separated from other eigenvalues, can be computed very quickly while eigenvalues that are tightly clustered together require more iterations or convergence may even fail. While advanced methods allow to compute all eigenvalues up to machine precision, it may not always be efficient to do so. Therefore, these methods are primarily used to compute partial eigendecomposition where only a subset $k \ll N$ of eigenpairs is obtained.

Among the fastest and most reliable methods are implicitly restarted variations of the Arnoldi (IRAM, Arnoldi, 1951, Sorensen, 1992) or Lanczos (IRLM, Calvetti et al., 1994, Lanczos, 1950) iterations. The latter applies to symmetric eigenproblems as used in this work and is explained further below.

Let i be the number of iterations the implicitly restarted Lanczos method requires to converge to a subset of eigenvalues on matrix $\mathbf{A}$. A partial eigendecomposition recovering k eigenpairs requires a worst-case estimate of

$$i\left[\rho cN + (6k+9)\rho N + 4\rho^2 N + 2k^2 N + O((k+\rho)^3)\right]$$

arithmetic operations (see Bai et al., 2000, Section 4.4.5). The number of extra Lanczos steps ρ can be assumed $O(k)$ and c is a sparsity factor of $\mathbf{A}$. For a dense matrix $c = 2N$ and for a sparse matrix with only $O(N)$ non-zero entries, the factor becomes a constant $0 < c \ll N$.

The number of iterations i cannot be predicted because it depends strongly on the distribution of the eigenvalues of $\mathbf{A}$. Since no strong assumptions about $\mathbf{A}$ can be made, i can differ much even when k, c, and ρ are the same. Therefore, exact computational cost cannot be predicted in general. It can however be compared to other eigensolvers in the limit of N. Under the assumption that $\rho = k$, the cost of each iteration is

$$ckN + 9kN + 12k^2N + O(8k^3). \tag{11}$$

When k and N are independent and $\mathbf{A}$ is dense, the cost is bounded by $O(kN^2)$ as $N \to \infty$. Furthermore, when $\mathbf{A}$ is sparse, the bound is only $O(k^2N)$. Therefore, IRLM can take great advantage of matrix sparsity and when $k \ll N$ its complexity is significantly lower.

To summarize, direct methods are used for the full eigendecomposition of small eigenproblems that can be stored in computer memory. Iterative methods do not require access to the full matrix and are best suited to compute a partial eigendecomposition recovering only a subset of k eigenvalues and eigenvectors. Direct methods have complexity $O(N^3)$ and iterative method $O(kN^2)$ for dense matrices and $O(k^2N)$ for sparse matrices. When $k = N$, iterative methods offer no time complexity advantage.

2.5 Eigenpair Perturbation Theory

The computation of eigenpairs of large matrices is a complex problem. In eigenvalue perturbation theory the setting is typically that a matrix under study $\mathbf{M}$ is almost the same as another matrix $\mathbf{M}^{(0)}$ for which the eigenpairs are known. Instead of solving the eigendecomposition for $\mathbf{M}$, eigenvalue perturbation theory describes the relationship between the eigenpairs of both matrices in terms of their difference $\mathbf{\Delta M}$. The idea is that this relationship enables a good approximation of the desired eigenpairs and that it is easier to compute. Eigenvalue perturbation theory is a major part of matrix perturbation theory where a setting as described above is typically expressed with the relationship

$$\mathbf{M} = \mathbf{M}^{(0)} + \mathbf{\Delta M},$$

where $\mathbf{M}^{(0)}$ is called the *unperturbed* matrix, $\Delta\mathbf{M}$ is the *perturbation*, and $\mathbf{M}$ is called the *perturbed* matrix. Typical problem fields that are approached with matrix perturbation theory are:

Numerical Precision Computers cannot represent real numbers with arbitrary precision and therefore numerical computations can introduce small rounding errors. It is possible to show the sensitivity of a computation to rounding errors by comparing an analytical result $\mathbf{M}^{(0)}$ to a result $\mathbf{M}$ contaminated by rounding errors. Here the perturbation $\mathbf{\Delta M}$ is the rounding errors.

Stochastic Approximations Many algorithms exploit randomness. For example, when the input is $\mathbf{M}^{(0)}$ but only the sample $\mathbf{M}$ is used for the computation, matrix perturbation theory can be used to determine requirements for valid results. Here the difference between the sample and the input constitutes the perturbation $\mathbf{\Delta M}$.

Noisy Observations When a theory about some idealized matrix $\mathbf{M}^{(0)}$ is used on input $\mathbf{M}$ contaminated by random noise ($\mathbf{\Delta M}$). This allows to map observed noisy data to some hidden or theorized process.

This work adopts an interpretation of the link prediction problem in terms of perturbation theory to approximate the eigenpairs of a *perturbed* adjacency matrix $\mathbf{A}$ based on the known eigenpairs of an *unperturbed* adjacency matrix $\mathbf{A}^r$. The *perturbation* is defined as $\Delta\mathbf{A} = \mathbf{A} - \mathbf{A}^r$ and $\mathbf{A}$ is real and symmetric as defined in Section 2.3. It follows that $\mathbf{A}^r$ and $\Delta\mathbf{A}$ are all real and symmetric.

Let ϵ be a perturbation parameter. Then the perturbed matrix can be written as

$$\mathbf{A} = \mathbf{A}^r + \epsilon\Delta\mathbf{A}, \tag{12}$$

with $0 \leq \epsilon \leq 1$. As ϵ approaches zero, the perturbed matrix becomes almost the same as the unperturbed matrix and eventually converges to $\mathbf{A}^r$. More importantly, Rellich and Berkowitz (1969) showed that the eigenpairs (λ_i, u_i) of $\mathbf{A}$ depend on ϵ and converge to the eigenpairs (λ_i^r, u_i^r) of $\mathbf{A}^r$ as ϵ approaches zero and the power series' for the perturbed eigenvalues

$$\lambda_i = \lambda_i^r + \sum_{j=1}^{\infty} \epsilon^j \lambda_i^{(j)} \tag{13}$$

and the perturbed eigenvectors

$$u_i = u_i^r + \sum_{j=1}^{\infty} \epsilon^j u_i^{(j)} \tag{14}$$

satisfy the eigenvalue problem (1) for a sufficiently small ϵ (Rellich and Berkowitz, 1969, Chapter 30, Theorem 1). Note that the notation for ϵ^j means ϵ raised to the power of j while (j) is used as an index.

2.5.1 First-Order Approximation of Eigenpair Perturbation

Rellich and Berkowitz' theorem can be used to approximate perturbed eigenpairs based on unperturbed eigenpairs that are known. In the remainder of this section a first-order approximation of the perturbed eigenpairs is derived assuming that all solutions (λ_i^r, u_i^r) to the eigenvalue problem of the unperturbed matrix

$$\mathbf{A}^r u^r = \lambda^r u^r \tag{15}$$

are known while the expansion terms $\lambda_i^{(j)}$ and $u_i^{(j)}$ in (13) and (14) remain to be determined. They are summed by order of ϵ and with increasing order the terms become smaller. A first-order approximation of the perturbed eigenpairs is obtained by truncating the series after the largest term, i.e. using only $\lambda_i^{(1)}$ and $u_i^{(1)}$ of order ϵ and ignoring the terms of higher order (i.e. ϵ^2, ϵ^3, etc.).

By substituting (12), (13), and (14) into (1) and truncating the power series at order ϵ, the first-order approximation for the eigenvalue equation of the perturbed matrix can be written as

$$(\mathbf{A}^r + \epsilon\boldsymbol{\Delta}\mathbf{A})(u_i^r + \epsilon u_i^{(1)}) = (\lambda_i^r + \epsilon\lambda_i^{(1)})(u_i^r + \epsilon u_i^{(1)}). \tag{16}$$

This equation can be expanded to

$$\mathbf{A}^r u_i^r + \epsilon(\mathbf{A}^r u_i^{(1)} + \boldsymbol{\Delta}\mathbf{A} u_i^r) + O(\epsilon^2) = \lambda_i^r u_i^r + \epsilon(\lambda_i^r u_i^{(1)} + \lambda_i^{(1)} u_i^r) + O(\epsilon^2),$$

where the second-order terms $O(\epsilon^2)$ can be ignored again. Using (15) we can simplify the equation to

$$\mathbf{A}^r u_i^{(1)} + \boldsymbol{\Delta}\mathbf{A} u_i^r = \lambda_i^r u_i^{(1)} + \lambda_i^{(1)} u_i^r. \tag{17}$$

The first-order perturbation of the eigenvalues $\lambda_i^{(1)}$ can be derived from (17) by projection on the set of orthonormal basis vectors. That is, expressed in terms of the unperturbed eigenvectors u_i^r instead of the unknown first-order perturbation vectors $u_i^{(1)}$:

$$u_i^{(1)} = \sum_{j=1}^{N} a_{ij} u_j^r. \tag{18}$$

Consider the orthonormality of the unperturbed eigenvectors, i.e. $(u_k^r)^\top u_j^r = \delta_{kj}$, and the equality in (15). Substituting (18) into (17) and left multiplying by $(u_k^r)^\top$ results in

$$a_{kk}\lambda_k^r + (u_k^r)^\top \boldsymbol{\Delta}\mathbf{A} u_i^r = a_{kk}\lambda_i^r + \lambda_i^{(1)}(u_k^r)^\top u_i^r. \tag{19}$$

For $k = i$, Equation (19) becomes

$$a_{ii}\lambda_i^r + (u_i^r)^\top \boldsymbol{\Delta}\mathbf{A} u_i^r = a_{ii}\lambda_i^r + \lambda_i^{(1)}. \tag{20}$$

Solving (20) for $\lambda_i^{(1)}$ gives the expression for the first-order eigen*value* perturbation

$$\lambda_i^{(1)} = (u_i^r)^\top \boldsymbol{\Delta}\mathbf{A} u_i^r. \tag{21}$$

The first-order eigen*vector* perturbation $u_i^{(1)}$ is obtained by solving (19) for the coefficients a_{kk} and then the vectors can be obtained from (18).

First, consider Equation (19) for the case $k \neq i$:

$$a_{kk}\lambda_k^r + (u_k^r)^\top \boldsymbol{\Delta}\mathbf{A} u_i^r = a_{kk}\lambda_i^r$$

which can be solved for a_{kk}

$$a_{kk} = \frac{(u_k^r)^\top \boldsymbol{\Delta}\mathbf{A} u_i^r}{\lambda_i^r - \lambda_k^r}, \quad k \neq i. \tag{22}$$

Inserting (22) into (18) gives the expression for the first-order eigenvector perturbation

$$u_i^{(1)} = \sum_{k=1\neq i}^{N} \frac{(u_k^r)^\top \boldsymbol{\Delta}\mathbf{A} u_i^r}{\lambda_i^r - \lambda_k^r} u_k^r. \tag{23}$$

However, for $k = i$, Equation (22) is undefined. This case has been considered in Equation (20) which can be simplified to

$$(u_i^r)^\top \mathbf{\Delta A} u_i^r = \lambda_i^{(1)}.$$

The meaning of this equation is that for $k = i$ the first-order perturbation is fully explained by the first-order eigenvalue perturbation and unaffected by u_i^r. This is confirmed in Champagne (1994) by showing that u_i^r is orthogonal to $u_i^{(1)}$. Since there are no additional constraints the coefficient can be chosen as $a_{ii} = 0$.

To improve notation and readability in the remainder of this work, the first-order perturbation terms are written as $\Delta\lambda_i$ instead of $\lambda_i^{(1)}$ for eigenvalue perturbation and Δu_i instead of $u_i^{(1)}$ for eigenvector perturbation.

2.6 Chebyshev Polynomial Approximation Theory

This section introduces the theory of approximating functions as polynomial expansions. In particular, the polynomials used here are Chebyshev polynomials which are covered in many textbooks, for example, Mason and Handscomb (2002) and Trefethen (2013). The discussion below refrains from proofs and derivations but should provide all necessary definitions to understand all elements of this theory that find use in Chapter 6 of this dissertation. Furthermore, the reader can find direct references to the relevant content in theoretical works.

Any continuous function defined on a closed interval can be approximated by a polynomial with arbitrary precision (Weierstrass, 1885). A particular family of polynomials that can be used for such approximations are Chebyshev polynomials. Introduced in the nineteenth century by the Russian mathematician P.L. Chebyshev, they have become an established topic of approximation theory and indispensable for the numerical approximation of functions on intervals. This section provides only a minimal overview of the most important definitions of Chebyshev polynomials. For the most part, proofs and derivations are referenced. All theory covered below is discussed in detail in Mason and Handscomb (2002), which will be referenced as MH (2002) below for brevity.

Function approximations with Chebyshev polynomials have several desirable properties. In particular, the approximation error exhibits the "minimax property" which is also called equioscillation theorem (MH, 2002, Lemma 3.6). It can be paraphrased by stating that the error is distributed over the approximation interval and alternating between two extrema with opposed sign. Furthermore, Chebyshev polynomials are a family of orthogonal polynomials. Approximations are therefore optimal in the $\mathcal{L}_2$ norm (MH, 2002, Theorem 4.1).

2.6.1 Chebyshev Polynomials

The trigonometric definition of Chebyshev polynomials for any x in $[-1, 1]$ is as follows. When

$$x = \cos\theta,$$

the Chebyshev polynomial of the first kind of degree j is denoted T_j and defined as

$$T_j(x) = \cos j\theta. \tag{24}$$

It follows that $T_j(x)$ is in the interval $[-1, 1]$. The polynomials of degree 0 and 1 can be immediately derived as

$$T_0(x) = 1 \qquad T_1(x) = x.$$

Instead of deriving polynomials from (24), all remaining polynomials can be obtained using the following recurrence relation (MH, 2002, Section 1.2.1),

$$T_j(x) = 2xT_{j-1}(x) - T_{j-2}(x) \quad \text{for } j \geq 2. \tag{25}$$

The computational benefits arising from the use of a Chebyshev expansion are rooted, partially, in this recurrence relationship.

The locations of the roots x_k can be derived from (24). Specifically, the zeros must occur where $\cos j\theta = 0$ for θ in $[0, \pi]$. Therefore, every Chebyshev polynomial T_j of degree j has its zero points in $[-1, 1]$ at

$$x_k = \cos\left(\frac{\pi(k - \frac{1}{2})}{j}\right) \qquad k = 1, \ldots, j. \tag{26}$$

Figure 2.3 displays the first few Chebyshev polynomials. It can be seen that the polynomials $T_j(x)$ have j zeros points.

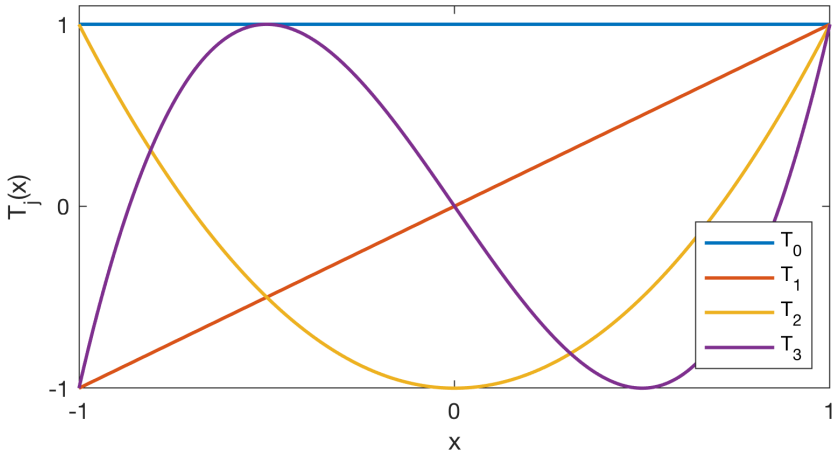


Fig. 2.3: *The Chebyshev polynomials of degree* 0 *to* 3.

2.6.2 Function Approximation with Chebyshev Polynomials

The following definition from Trefethen (2013, Theorem 3.1) will be used throughout this section. Let h be a continuous real function that is Lipschitz continuous on the interval

$[-1, 1]$. Then, it can be represented as an absolutely and uniformly convergent Chebyshev series

$$h(x) = \sum_{j=0}^{\infty} c_j T_j(x), \tag{27}$$

where c_j are the Chebyshev series coefficients defined by the integral

$$c_j = \frac{2}{\pi} \int_{-1}^{1} \frac{h(x) T_j(x)}{\sqrt{1 - x^2}} dx \tag{28}$$

Importantly, the coefficient c_0 is defined as (28) but halved (see MH, 2002, Eq. 4.11a and Eq. 4.11b). Because this is inconvenient, an equivalent but more convenient definition of (27) is

$$h(x) = \frac{1}{2} c_0 T_0(x) + \sum_{j=1}^{\infty} c_j T_j(x), \tag{29}$$

which allows to define the coefficients as in (28) for all $j \geq 0$ (MH, 2002, Eq. 5.7).

In general, we are interested in approximating h with a finite polynomial series. This can be obtained by truncation of the Chebysev series to degree m. This method is sometimes also called projection of the series. The approximation h_m of h by truncation after degree m is

$$h_m(x) = \frac{1}{2} c_0 T_0(x) + \sum_{j=1}^{m} c_j T_j(x). \tag{30}$$

A visual example of a truncated Chebyshev series approximation is shown in Figure 2.4 (blue curve).

2.6.3 Chebyshev Polynomials on the Interval $[a, b]$

Chebyshev polynomials are defined for the variable x in the interval $[-1, 1]$ but many functions of interest are defined on arbitrary, finite intervals $[a, b]$. As long as the function is Lipschitz continuous on $[a, b]$, all properties derived above hold. The Chebyshev polynomials can be *shifted* from $[-1, 1]$ to $[a, b]$ using a simple linear transformation (see MH, 2002, Section 1.3). For convenience, let the transformed variable be denoted y and defined

$$y \coloneqq \frac{x - \alpha_1}{\alpha_2}, \tag{31}$$

with

$$\alpha_1 = \frac{b + a}{2}$$
$$\alpha_2 = \frac{b - a}{2}.$$

The shifted Chebyshev polynomials of the first kind are defined as

$$\overline{T}_j(y) \coloneqq T_j\left(\frac{x - \alpha_1}{\alpha_2}\right). \tag{32}$$

2.6.4 Approximating a Step-Function and Jackson Smoothing

Chebyshev series approximation of functions with finite step discontinuities exhibit the so-called Gibbs phenomenon discovered by Wilbraham (1848). In Figure 2.4, it is demonstrated on a small example on the function $h(x) = sign(x)$ that has a discontinuity at $x = 0$. The blue curve is a truncated Chebyshev series approximation of h with degree $m = 90$. Strong oscillations occur at the discontinuity and the approximation "overshoots". This error has been shown to converge to a finite limit, that means, it does not become zero as the degree $m \to \infty$ (Carslaw, 1921).

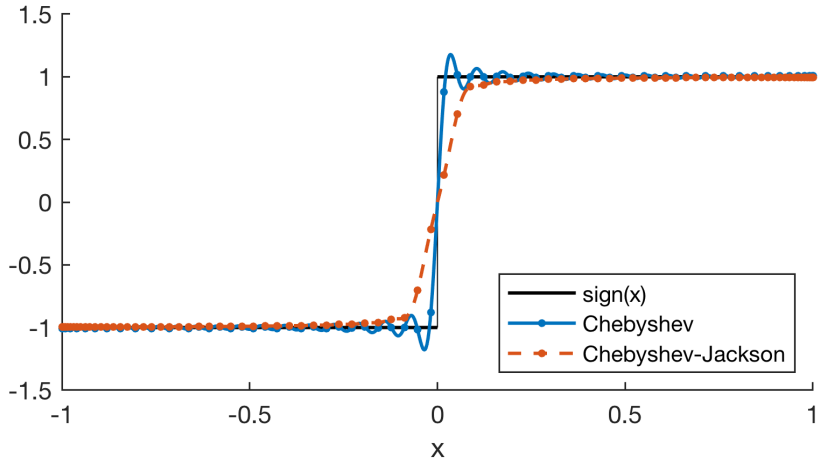


Fig. 2.4: *The $sign(x)$ function has a discontinuity at $x = 0$. The standard Chebyshev series approximation (blue) exhibits the Gibbs phenomenon at the discontinuity. The dashed, red curve is the smoothed version using Jackson coefficients.*

A solution to the Gibbs phenomenon is to use damping factors. These damping factors are based on the Jackson approximation and were proposed by Silver et al. (1996) and Schofield et al. (2012). Each Chebyshev expansion coefficient c_j has an associated damping factor called a Jackson coefficient g_j. The truncated Chebyshev-Jackson series becomes

$$h_m(x) = \frac{1}{2} g_0 c_0 T_0(x) + \sum_{j=1}^{m} g_j c_j T_j(x). \tag{33}$$

The corresponding Jackson coefficients g_j as defined in Di Napoli et al. (2016) are:

$$g_j = \frac{\sin(j+1)\beta}{(m+2)\sin\beta} + \left(1 - \frac{j+1}{m+2}\right)\cos(j\beta) \tag{34}$$
$$\beta = \frac{\pi}{m+2}.$$

The dashed, red curve in Figure 2.4 demonstrates the smoothing effect of the Jackson coefficients.

2.7 Classifier Evaluation

The link-existence estimators $\langle \hat{\mathbf{A}} \rangle$ obtained from SPM link prediction (see Section 3.3) are used to map each *unobserved edge* of a graph to one of two sets; *missing edges* and *non-existing edges*. Correctly labeling missing edges is considered a benefit and falsely assigning a missing label to non-existing edges is considered a loss. The quality of an estimator is assessed by its ability to distinguish between the two classes. This makes SPM link prediction a classical two-class classification problem.

A classification problem is given by a set of *instances* which can be thought of as observations that require classification. A classification *model* defines how to assign a class label to each instance. A two-class classification model makes binary class decisions; the classification is answering the question "does instance i belong to class c?". The class label representing a "yes" answer is called *positive* class (p) and the other class is called *negative* class (n). For evaluation purposes, the set of all instances is split into a *training set* and a *test set*. A classification model tunes an arbitrary mechanism to assign labels to each instance in the training set. This yields a concrete instance of the classification model, a *classifier*, that can then be evaluated on the test set of instances.

The output of a classifier is an assignment of a predicted class label p' or n' to each instance i in the test set. Each classification of an instance i has four possible outcomes. When i is classified as positive (p') and the true class of i is also positive (p), then the outcome is a *true positive* (TP). In case the true class is negative (n), it becomes a *false positive* (FP). When i is classified as negative (n'), it can either be a *false negative* (FN) or a *true negative* (TN) depending on whether the true class of i is negative (n) or positive (p). Counting each such outcome separately creates a *confusion matrix* (also called the contingency table) as shown in Figure 2.5.

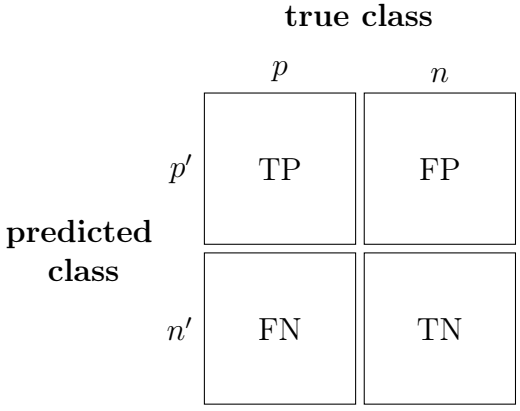


Fig. 2.5: *A confusion matrix. Different classifier performance metrics are defined based on the confusion matrix.*

Many performance metrics for classifiers evaluation are defined based on the confusion matrix. Precision is the ratio of how many positive predictions are correct:

$$\text{Precission} = \frac{TP}{TP + FP}. \tag{35}$$

Accuracy is the ratio of all correct predictions (positive and negative) to the total of all predictions made.

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + TN + FP}. \tag{36}$$

Note that the labeling of classes as positive and negative is arbitrary and can be changed depending on what is being optimized.

2.7.1 ROC Space

The receiver operating characteristic (ROC) curve enables the evaluation of classifiers based on a benefit-cost trade-off assumption implied by the definition of positive and negative classes. A true positive classification is associated with a benefit and a false positive classification is associated with a loss. The ROC curve has first been used to analyze this trade-off in signal detection theory (Egan, 1975) but has since spread to many other domains.

ROC space is defined by the true positive rate (TPR) on the y-axis and the false positive rate (FPR) on the x-axis. These are defined as

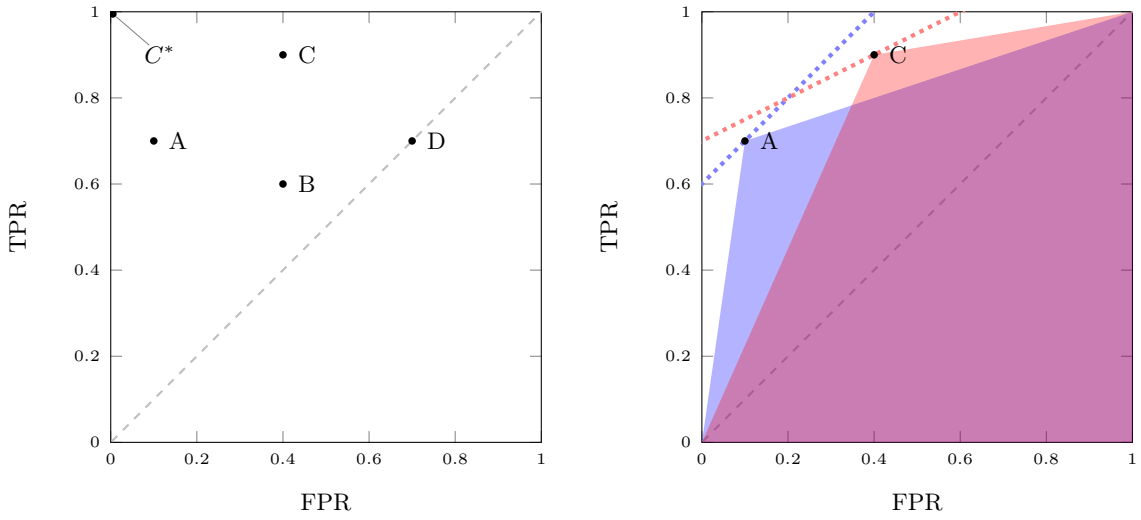
$$\text{TPR} = \frac{TP}{TP + FN} \quad ,$$

that is the ratio of all positives that were predicted correctly, and as

$$\text{FPR} = \frac{FP}{FP + TN} \quad ,$$

the ratio of all negatives that were misclassified as positive.

Any classifier emitting two-class classifications can be placed on a point in ROC space as shown in Figure 2.6a. The perfect classifier C^* labels all positives correctly and has TPR = 1.0 and FPR = 0.0. Therefore, it is located at point $(0, 1)$ in ROC space. Classifiers move towards the origin in ROC space as they are hesitant to commit to positive classifications. They can still be very precise but they are "conservative" because such classifiers avoid the cost of false positives and, as a consequence, issue many false negative classifications. In $(0, 0)$ a classifier never classifies instances as positive. As a classier is located towards the top right in ROC space it is more "liberal", risking the cost of false positives to increase recall. In the extreme of point $(1, 1)$ it classifies every instance as positive. Classifiers located anywhere on the diagonal make uninformed decisions equivalent to random guessing. They have equal TPR and FPS, that means, they cannot distinguish between true positives and false positives. Note that the ROC space is symmetric about the diagonal and any classifier below the diagonal can be mirrored on the diagonal by inverting its classifications.



(a) *Example ROC space with different classifiers.* **(b)** *Comparing classifiers in ROC space by area under curve or iso-performance lines.*

Fig. 2.6

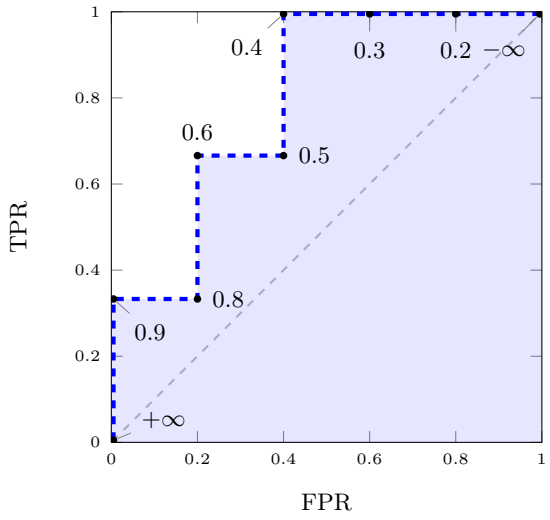
The ROC space is used to evaluate the trade-off between TPR (benefit) and FPR (cost). In Figure 2.6a classifier A $(0.1, 0.7)$ can be considered better than B $(0.4, 0.6)$ as it has higher TPR and lower FPR. And B is better than D $(0.7, 0.7)$; even though it has lower TPR, the overall trade-off is better. D has both rates equal which means it does not distinguish between true positives and false positives. This is consistent with classifying any instance as positive at a constant rate; which is uninformed decision making.

In practice, these comparisons are made by computing a metric that quantifies the performance of the classifier. In most ROC evaluations this metric is the area under the ROC curve (AUC). It is depicted in Figure 2.6b for A (blue area) and C (red area). One can see visually that A's area is larger. The next section introduces ROC curves. The methods for classifier comparison are discussed in more detail in Section 2.7.3.

2.7.2 ROC Curves

In most cases, classifiers compute internal scores or probabilities for each instance and use this score to make a classification decisions by comparing it to a *threshold* value that represents a decision boundary. Changing this threshold, changes the confusion matrix that the classifier produces on a given test set and this change moves the classifier to a different point in ROC space. Measuring a classifier's performance at every possible threshold yields a curve in ROC space.

In Figure 2.7, a single classifier instance is evaluated at different thresholds. The table on the right lists the true class for each instance in the test set (second columns) and the computed scores in descending order (third column). The decision rule for the classifier is that every instance with score larger or equal to the threshold is assigned to the positive class. All other instances are assigned to the negative class.



Instance	True Class	Score
1	p	0.9
2	n	0.8
3	p	0.6
4	n	0.5
5	p	0.4
6	n	0.3
7	n	0.2
8	n	0.1

Fig. 2.7: *Example curve in ROC space for a single classifier at different thresholds.*

The ROC curve is built up by testing the classifier at different thresholds. The initial threshold is $+\infty$ and the classifier maps all inputs to the negative class because every score is smaller than infinity. The point for this threshold is always $(0, 0)$. Equating the threshold to the largest score in the table, 0.9, maps instance 1 to the positive class. The classification for instance 1 happens to be a true positive. All other instances are below the threshold and therefore mapped to the negative class. This results in only one of three positive instance being classified correctly, that is, $\text{TPR} = \frac{1}{1+2} = 0.333$. The FPR is still 0 and the classifier moves to $(0, 0.333)$ in ROC space for a threshold of 0.9. This point is shown in Figure 2.7. At threshold 0.8, instance 2 is misclassified as positive and $\text{FPR} = \frac{1}{1+4} = 0.2$ with the same TPR. The classifier moves to point $(0.2, 0.333)$ in ROC space. Proceeding in this manner trough all thresholds until $-\infty$, which is always at point $(1.0, 1.0)$, produces the ROC curve (dashed blue line) in Figure 2.7.

Implementations of algorithms that compute the ROC curve are widely available. Refer to Fawcett (2004) for templates if a custom implementation is required.

2.7.3 Comparing Classifier Performance in ROC Space

ROC curves have two desirable properties. First, the scores produced by the classifier need not be calibrated or normalized into specific ranges. The ROC curve depicts the ability of the classifier to discriminate between positive and negative instances of a class. When all positive instances are ranked before negative instances, the ROC curve will correctly indicate a perfect performance. This is characterized by the fact that it never moves towards the right in ROC space before reaching $(0, 1)$. Otherwise, the ROC curve starts to turn away from $(0, 1)$ before reaching it because the FPR increases.

Second, the ROC curve is insensitive to unbalanced classes because the ROC score remains the same as long as the classifier's ability to distinguish between the two classes does not change. For example, simply duplicating all negative instances in the table in

Figure 2.7 does not change the ROC curve at all because the same FPR ratios apply at every threshold and the TPR is unaffected entirely. In contrast, the precision metric will be lower at most thresholds because of the changed distribution between total TP and FP classifications even though the classifier's ability to distinguish the classes has not changed at all.

Area Under Curve (AUC): When evaluating classifier performance it is often desired to quantify their performance with a single numeric value that can be trivially compared and analyzed with descriptive statistics.

The common metric to compare the expected performance of classifiers in ROC space is the area under the ROC curve (AUC, Hanley and McNeil, 1983). In Figure 2.6b, the AUC for two classifiers with fixed thresholds are shown by the red and blue shaded areas. A's area is larger and indeed, it has a better ratio of TPR to FPR which signifies a better cost-benefit trade-off. However, notice the dashed lines with differing slopes. The blue line passing through A has a slope of 1, and marks the line of equal performance when FPR and TPR are considered equally important. A is better because its line is closer to $(0, 1)$. However, the trade-off between TPR and FPR can be weighed differently. When a larger TPR is prioritized and weighed double, the ratio is $TPR/(2 * FPR)$ and the equal performance line has a slope of 0.5 (dashed, red). In this case, C is better than A. This shows that such comparisons are always circumstantial and that a one-dimensional value such that the AUC is necessarily limited in what it can express. Nevertheless, in most comparisons the AUC is a good choice (Fawcett, 2004).

For a generic ROC curve, the AUC is shown in Figure 2.7 shaded in blue. The ROC space is defined in the unit square. Therefore, the maximum AUC value is 1 and the minimum is 0. In practice, the minimum value is 0.5 which represents random guessing. The AUC represents an average performance of a classifier at all evaluated thresholds. Importantly, it is equivalent to the probability that random positive instances are ranked higher than random negative instances (Hanley and McNeil, 1983). An efficient way to calculate the AUC can be derived from this probabilistic interpretation.

Given two unordered lists; S_{tp} with scores of individual true positive instances and S_{tn} with scores of individual true negative instances. Let C be a set of pairs (s_{tp}, s_{tn}) such that each s_{tp} is chosen uniformly at random from S_{tp} and each s_{tn} is chosen uniformly at random from S_{tn}. Suppose the set $C^{(+)} \coloneqq \{(s_{tp}, s_{tn}) \in C : s_{tp} > s_{tn}\}$ contains all chosen pairs where the true positive instance is ranked higher than the true negative instance and set $C^{(=)} \coloneqq \{(s_{tp}, s_{tn}) \in C : s_{tp} = s_{tn}\}$ contains all pairs where their score is equal. The AUC is approximated by:

$$\text{AUC} = \frac{|C^{(+)}| + 0.5|C^{(=)}|}{|C|}. \tag{37}$$

This method is computationally efficient and all AUC scores reported in this dissertation are obtained using Equation (37). The AUC represents the ability of a classifier to discriminate between positive and negative instances across the entire ROC space. Depending on the use-case, some regions may be more interesting than others. In this case

the ROC curve rather than the AUC needs to be analyzed to determine the performance of a classifier in the regions of interest.

Averaging ROC Curves: In this work SPM classifiers are tested many times on different test-training splits. To report the performance of an SPM classifier with a given set of parameters on a given graph, all corresponding ROC curves are averaged.

Curve averaging is done with the *vertical averaging* (VA) method first used by Provost and Fawcett (1998). This method samples from a set of ROC curves at defined sample points along the x-axis (FPR). Each sample point thus consists of a set of TPR values which can be averaged to obtain an average TPR point in ROC space. Let the set of ROC curves to sample be R where each curve $r \in R$ is a continuous function $r : \mathbb{R} \to \mathbb{R}$ such that $r(x)$ is the TPR at FPR $= x$ with $0 \leq x \leq 1$. Furthermore, let $S \in \mathbb{Z}^+$ be the number of samples to average. The average TPR value at sampling point $s \in \{1, \ldots, S\}$ is:

$$\mu_s = \frac{1}{|R|} \sum_{r \in R} r(s).$$

Therefore, the average ROC curve can be defined as a function $\bar{r} : \mathbb{R} \to \mathbb{R}$ such that $\bar{r}(s) = \mu_s$.

VA is implemented in Algorithm 2.1. The input is the number of samples S and R; the set of ROC curves to average. A curve is defined by a pair (f, t), where f and t are vectors of equal length containing the x-axis (FPR) and y-axis (TPR) coordinates respectively at each evaluated threshold. The algorithm samples the ROC curves at uniformly distributed FPR values. The sample points are generated on Line 3. Each ROC curve in R is evaluated at each sampling point. The ROC curves are discrete. Therefore, it can occur that no TPR value exists for a sampling point. In this case, the TPR is linearly interpolated between the two closest neighboring points (Line 6). The (interpolated) samples from each curve are collected and averaged (Line 9). The output is an average ROC curve consisting of S points in ROC space (Line 10).

Algorithm 2.1 Vertical Averaging of ROC Curves

Input: R, S

```
1:  n_curves ← |R|
2:  i_tprs ← EMPTY(S, n_curves)
3:  mean_fpr ← LINSPACE(0.0, 1.0, S)        ▷ Vector of S uniformly distributed real values ∈ [0, 1[
4:  for i ← 1; i ≤ n_curves; i ← i + 1 do
5:      (f, t) ← R(i)
6:      i_tprs(i) ← INTERPOLATE(mean_fpr, f, t)
7:      i_tprs(i, 0) ← 0.0                  ▷ The curve always starts at point (0.0, 0.0)
8:      i_tprs(i, S) ← 1.0                  ▷ The curve always ends in point (1.0, 1.0)
9:  mean_tpr ← ROWAVERAGE(i_tprs)           ▷ The interpolated TPR vectors constitute the columns.
10: return (mean_fpr, mean_tpr)
```

Macskassy and Provost (2004) propose to calculate confidence intervals for ROC curves obtained by VA under the assumption of binomial distribution. Every TPR is a ratio

of successes. Therefore, each average TPR is a success probability (or proportion) and a binomial distribution can be assumed. The normal approximation method for the binomial approximation interval is defined as

$$\mu_s \pm z\sqrt{\frac{\mu_s(1-\mu_s)}{|R|}}$$

for $s \in \{1, \ldots, S\}$ and z, the quantile of the standard normal distribution corresponding to the two-tailed desired significance level (z-score). For a 95% confidence interval, $z \approx 1.96$.

Chapter 3

Spectral Link Prediction on Graphs

This chapter presents an application of matrix perturbation theory: the structural consistency index of a graph which is a quantification of edge predictability in graphs. Furthermore, this index can be extended into a link prediction method. Much of this chapter discusses the properties of this link prediction approach to provide a basis upon which the extensions developed in Chapters 5 and 6 can be evaluated.

3.1 The Link Prediction Problem

Graphs are static representations of interactions in complex systems. As most of these systems are not static and environments change, their graph representations have to adapt to remain useful. A graph describing the state of evolving interactions is destined to become an inaccurate representation as time progresses and the represented system changes. Even when a graph is updated continuously it may not be free from errors. As they are human concepts imposed upon models and data, graphs cannot be perfect and suffer from errors and incompleteness. Considering the enormous size and complexity that many systems described by graphs display, it is hardly imaginable that they can be captured with complete accuracy.

Link prediction addresses these issues where they concern the interaction between vertices, that is; graph edges (links). Edges often capture the most relevant dynamics of a represented system. Consider having obtained a list of all airports in the world without any information about flights that connect them. It would be difficult to plan travels or model the transmission of diseases by airline passengers.

Gathering information on edges that relate to any form of physical interactions can be difficult as such edges are not always trivially observable and may be hidden on purpose. The discovery or verification of links can demand significant investments of labor, money, and computation. Having reliable information about the likelihood and strength of potential interaction represents a valuable commodity in such scenarios and translates to better resource allocation options. Even in virtual systems links represent value. Online stores benefit greatly when customers perceive product recommendations as relevant. State-of-the-art recommender systems use a graph representation where customers and goods are modeled as vertices with edges indicating the likelihood of a purchase. Being able to recommend relevant goods based on past interactions of similar users can positively influence sales and customer satisfaction. In chemistry and biology, molecule interactions are modeled as graphs and researchers are interested in different interaction patterns. However, these patterns are difficult to identify as there can be billions of possibilities and some patterns are complex configurations of interacting vertices. Link prediction can help researchers by identifying potentially unknown interactions based on the presence

of previously discovered patterns in other molecules and therefore inform the design of promising experiments.

These are only some scenarios that may be of interest to the reader. A common element is that link prediction assigns scores or likelihoods to interactions (edges) in graphs based on evidence already present in these systems. Primarily, link prediction is concerned with interactions that have not yet been observed. They can be interpreted as future interactions that are likely to occur or as interactions that have always been present but not observed before. The first scenario is actual prediction while the second scenario describes a graph completion approach. Another problem that is often addressed by the same methods is the identification of spurious links, that is, edges that are likely erroneous observations or noise. An extensive survey of link prediction methods has been conducted in different articles by Liben-Nowell and Kleinberg (2007), Lü and Zhou (2011) and Wang and Liao (2014).

In this chapter, a link prediction method that uses the spectral features of a graph as evidence for the existence of unobserved interactions is described. The spectral features, or the eigenpairs, of a graph are fundamental properties describing the interactions of a graph adjacency matrix. As this information concerns the graph as a whole, this approach is sometimes called a *global method*. Most global methods are computationally costly but also powerful. Approaches that use only the information associated to individual vertices and their direct neighbors are called *local methods*. The state-of-the-art is such that local or quasi-local methods are successfully used on large graphs due to their relatively low complexity. But when evaluated on small graphs, global methods tend to perform at a significantly better accuracy. This thesis presents methods to make global link prediction applicable to significantly larger graphs with the aim to harness their power in a wider field of applications. These extensions are the subject of Chapters 5 and 6.

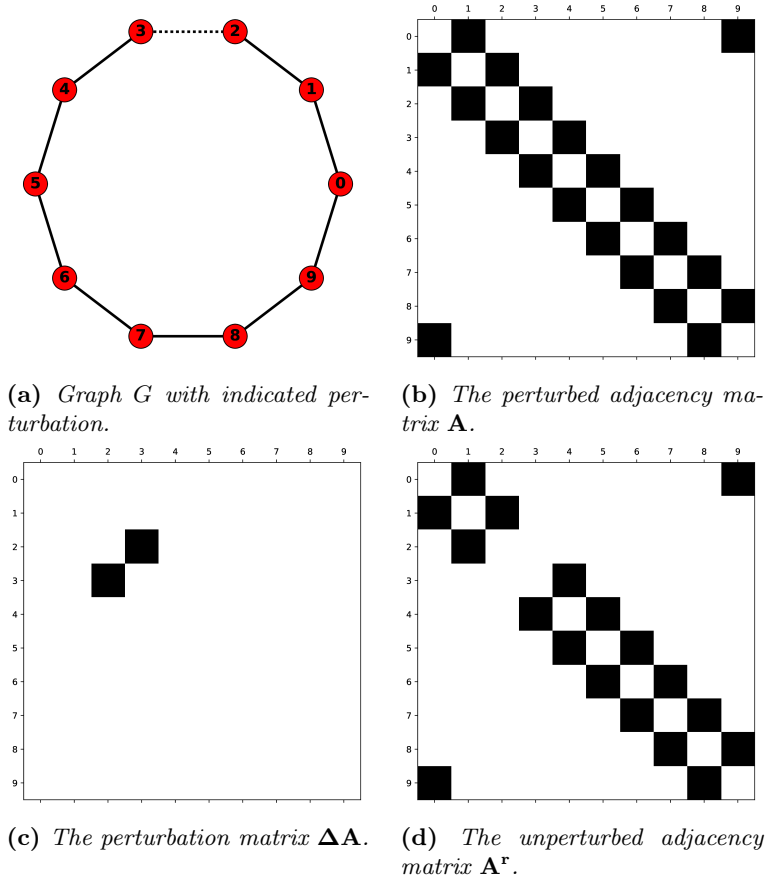
The remainder of this chapter is organized as follows. Section 3.2 defines the structural consistency index and analyzes some of its properties. SPM link prediction is defined in Section 3.3 and its computational aspects are analyzed in Section 3.4. The chapter concludes with a short assessment of SPM (Section 3.5).

3.2 The Structural Consistency of a Graph

Lü et al. (2015) introduced the *structural consistency* measure that quantifies the edge predictability of a graph. It measures the sensitivity of a graph's spectral properties to perturbations introduced by the addition (or deletion) of a small number of edges. When such a perturbation does not affect the graph structure significantly, then the unperturbed and perturbed graphs should be almost equal and the eigenpairs of one can be used to approximate the other. Intuitively, this should hold when the edges of a graph are created according to some regular pattern. Regularities allow predictions based on observed patterns while random structures are unpredictable. Figure 3.1a depicts an example of a very regular ring graph.

Formally, the structural consistency is defined for an adjacency matrix $\mathbf{A}$ (the perturbed matrix) and its corresponding graph $G(V, E)$. A small subset of edges ΔE is

selected as the perturbation set by choosing a fraction p of the edges in E uniformly at random. Removing ΔE from G yields the graph $G^r(V, E^r)$ with the same vertices as the G but with fewer edges. The corresponding adjacency matrix $\mathbf{A^r}$ (the unperturbed matrix) is square and symmetric, and can be diagonalized as in Equation (2). Furthermore, the perturbation set ΔE has a corresponding perturbation matrix $\mathbf{\Delta A} = \mathbf{A} - \mathbf{A^r}$. The example in Figure 3.1 illustrates above definitions.



(a) *Graph G with indicated perturbation.*

(b) *The perturbed adjacency matrix* $\mathbf{A}$.

(c) *The perturbation matrix* $\mathbf{\Delta A}$.

(d) *The unperturbed adjacency matrix* $\mathbf{A^r}$.

Fig. 3.1: *An example of a graph perturbation setting with associated matrices. (a) A ring graph with obvious structural pattern. The perturbation $\Delta E = \{(2, 3)\}$ is indicated by the dotted line in (a). The* perturbed *graph G includes edge $(2, 3)$ while the* unperturbed *graph G^r does not. (b) The adjacency matrix corresponding to G. (c) The matrix corresponding to the perturbation set ΔE. (d) The adjacency matrix corresponding to G^r.*

The setting introduced here is equivalent to a matrix perturbation problem as defined by Equation (12) in Section 2.5. The magnitude of the perturbation ΔE is controlled by parameter p. Lü et al. (2015) define ΔE to be a small perturbation. Here, the meaning of small is: sufficiently small for the perturbation theory defined in Section 2.5 to hold.

Consider $\mathbf{\Delta A}$ to be a perturbation to $\mathbf{A^r}$. The eigenpairs of $\mathbf{A}$ become $(\lambda_i^r + \Delta\lambda_i,\ u_i^r + \Delta u_i)$ for $i \in \{1, \ldots, N\}$. The Δ-terms denote the introduced pertur-

bation. The eigenfunction for $\mathbf{A}$ can now be rewritten as

$$(\mathbf{A^r} + \mathbf{\Delta A})(u_i^r + \Delta u_i) = (\lambda_i^r + \Delta\lambda_i)(u_i^r + \Delta u_i).$$

Lü et al. make the assumption that a small perturbation and does not change the eigenvectors significantly. Therefore, the eigenvector perturbation Δu_i is ignored and only a first-order approximation of the eigenvalue perturbation is considered:

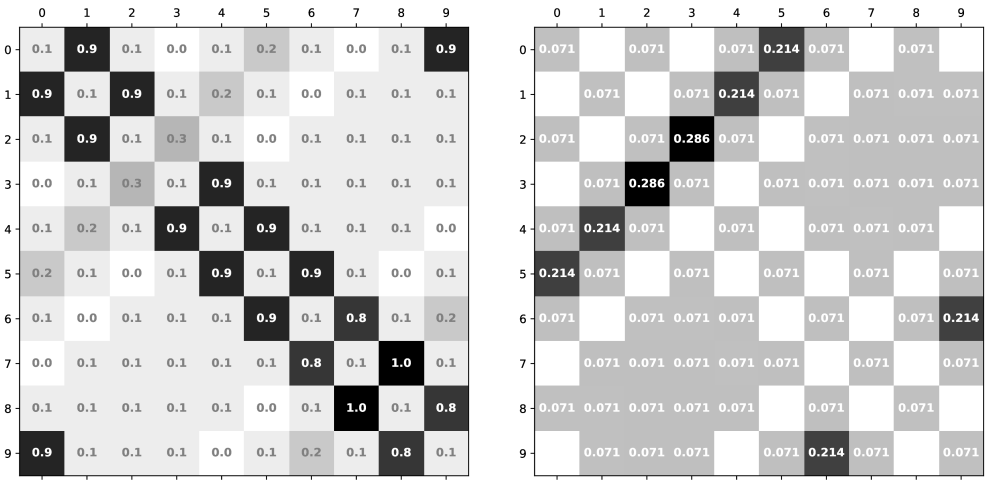
$$\Delta\lambda_i = (u_i^r)^\top \mathbf{\Delta A} u_i^r. \tag{38}$$

This term is derived as Equation (21) in Section 2.5.

Let matrix $\hat{\mathbf{A}}$ denote an approximation of the perturbed matrix $\mathbf{A}$. The elements $A_{ij} = 1$ if the edge $(v_i, v_j) \in E$ or $A_{ij} = 0$ otherwise. Each element $\hat{A}_{ij}$ is an approximation of the corresponding A_{ij}. The matrix $\hat{\mathbf{A}}$ is obtained via the eigendecomposition for the perturbed matrix as given by Equation (2). An approximation of the perturbed eigenvalues is obtained by addition of the perturbation term from Equation (38) to the known unperturbed eigenvalues:

$$\mathbf{A} \approx \hat{\mathbf{A}} = \sum_{i=1}^{N} (\lambda_i^r + \Delta\lambda_i) u_i^r (u_i^r)^\top. \tag{39}$$

The approximation $\hat{\mathbf{A}}$ does not recover the adjacency matrix $\mathbf{A}$ exactly due to the first-order truncation of the eigenvalue approximation and because the eigenvector perturbation is ignored. Nevertheless, when the eigenpair approximation is accurate any 0 or 1 valued element in $\mathbf{A^r}$ should have a corresponding element with value close to 0 or 1 respectively in $\hat{\mathbf{A}}$ (see Figures 3.1d and 3.2a). Crucially, the perturbed elements that are 0 in $\mathbf{A^r}$ but 1 in $\mathbf{A}$ will have a relatively large value (see $\hat{A}_{23}$ and $\hat{A}_{32}$ in Figures 3.2a and 3.2b). Algorithm 3.1 details the procedure to calculate $\hat{\mathbf{A}}$.



(a) *The first-order approximation* $\hat{\mathbf{A}}$. **(b)** *Heatmap of unobserved edges for* $\hat{\mathbf{A}}$.

Fig. 3.2: *The result of a first-order approximation for the example given in Figure 3.1. (a) The approximation of* $\mathbf{A}$ *computed with Equation* (39)*. (b) A heatmap showing the approximated values only for edges that do not exist in the unperturbed graph* $\mathbf{A^r}$*. This makes it easy to spot that the perturbed edges* $\hat{A}_{23}$ *and* $\hat{A}_{32}$ *score the highest among the unobserved edges.*

The structural consistency metric is defined as:

$$\sigma_c := \frac{|E^l \cap \Delta E|}{|\Delta E|}.$$

where E^l is a set containing $l = |\Delta E|$ *unobserved* edges. Given the unperturbed graph $G^r(V, E^r)$ any edge (v_i, v_j) is unobserved when $v_i \in V$ and $v_j \in V$ and $(v_i, v_j) \notin E^r$. Each unobserved edge (v_i, v_j) is ordered descending by its corresponding value $\hat{A}_{ij}$ in $\hat{\mathbf{A}}$ and only the first l unobserved edges constitute the set E^l. Note that the edges are undirected so $(v_i, v_j) \equiv (v_j, v_i)$. The structural consistency is the fraction of edges in this set that are also in the perturbation set. Ideally they are the same: $E^l = \Delta E$. In the example depicted in Figure 3.2 the elements $\hat{A}_{23}$ and $\hat{A}_{32}$ corresponding to ΔE (c.f. Figure 3.1c) have the largest value of all unobserved edges. This means the structural consistency is 1 for the example. For less obvious patterns encountered in data from real-world observations the accuracy is lower. Refer to Lü et al. (2015) for a rigorous evaluation of structural consistency.

It should be stressed that nothing in the above definitions is restricted to adjacency matrices that contain only values 0 and 1. The theory upon which the structural consistency is defined holds for any real and symmetric matrix.

3.2.1 Interpretation of Structural Consistency

When the eigenpairs of an adjacency matrix are called *structural properties* of its corresponding graph, then the structural consistency quantifies the precision at which the

removed edges are ranked at the top of a list of unobserved edges. The order of ranking these edges is determined by how well the structural properties of $\mathbf{A^r}$ can "explain" them. Since the adjacency matrix is used here, the eigenpairs "explain" the adjacency relationships between vertices of the corresponding graph.

Since the eigenvalue perturbation is approximated but the eigenvector perturbation is not, σ_c can be interpreted as a measure of similarity between the eigenvectors of $\mathbf{A}$ and $\mathbf{A^r}$. The mathematical meaning of eigenvectors is that the associated matrix acts like a scalar (the corresponding eigenvalue) on the eigenvector. Therefore, $\hat{\mathbf{A}}$ is an approximation of $\mathbf{A}$ by only scaling the unperturbed eigenvectors with different eigenvalues. When this produces a good match, then the structural properties captured by the eigenvectors of the unperturbed matrix are similar to the properties giving rise to the perturbed matrix. When the approximation is inaccurate, some important structural properties must have changed. Then the structural properties of $\mathbf{A^r}$, and specifically its eigenvectors, have become inconsistent with $\mathbf{A}$.

The next section investigates some conditions for eigenvectors to remain consistent under perturbation.

3.2.2 Conditions for Invariant Eigenvectors

The assumption that eigenvectors are not significantly affected by a perturbation when a matrix has high structural consistency is not formally justified in Lü et al. (2015). There is empirical evidence that the approach works in some settings and the authors' claim is that if graph G is "highly regular, the random removal ΔE will not sharply change the structure features" (Lü et al., 2015, p. 2326) without providing any theoretical justification. Indeed it would be very difficult provide a general guarantee as it is easy to construct synthetic graphs that are sensitive to small perturbations.

Guidance can be obtained by the classical Davis-Kahan theorem (1970). It provides a bound on the magnitude of the perturbation for symmetric matrices under symmetric perturbations. Let θ_i denote the angle between the eigenvectors u_i and u_i^r of $\mathbf{A}$ and $\mathbf{A^r}$ respectively. The sine of the angle between eigenvectors before and after perturbation measures the magnitude of their change in direction. The Davis-Kahan theorem bounds this quantity by

$$\sin(\theta_i) \leq \frac{\|\mathbf{\Delta A}\|}{\text{gap}_{\lambda_i}(\lambda^r)},$$

where $\text{gap}_{\lambda_i}(\lambda^r) \coloneqq \min\{|\lambda_j^r - \lambda_i| : j \neq i\}$ is the smallest distance between the corresponding perturbed eigenvalue λ_i and adjacent unperturbed eigenvalues λ_j^r. $\|\mathbf{\Delta A}\|$ denotes the spectral norm which is equal to its largest eigenvalue ($\lambda_1^{\Delta A}$) and quantifies the amplification power of $\mathbf{\Delta A}$.

The last statement can be interpreted better when the perturbation matrix is seen as an operator acting on some vector. Diagonalizing $\mathbf{\Delta A}$ as in Equation (2) and using Equation (6) to project a vector x onto the eigenbasis of the perturbation matrix yields

$$\mathbf{\Delta A}x = \mathbf{U}^{\Delta A}\mathbf{\Lambda}^{\Delta A}\hat{x}.$$

Recall that $\hat{x}(1)$ contains the length of x along the eigenvector corresponding to the largest eigenvalue (see Section 2.3.3). This vector represents the input direction that is amplified most, because it is multiplied by the largest eigenvalue. Thereafter, the result of this amplification is projected back into the original vector space. Without making assumptions about the direction of x, the exact effect is unknown. The maximal change to x that $\mathbf{\Delta A}$ can create occurs when x is exactly aligned with the largest eigenvector. Then x is scaled, or amplified, by $\lambda_1^{\Delta A}$.

A very similar relationship has been derived in Equation (23) where the magnitude of the dividend is related to $\|\mathbf{\Delta A}\|$ and the the divisor is equal to $\text{gap}_{\lambda_i^r}(\lambda^r)$. Therefore, the sensitivity of an eigenvector to perturbations is small when the following conditions are satisfied:

1. The perturbation matrix $\mathbf{\Delta A}$ has a small matrix norm or amplification power; and
2. The gap between adjacent eigenvalues is large before and after perturbation.

Regarding the first condition: since $\mathbf{A}$ is taken to be sparse with only non-negative entries, selecting a fraction p of its randomly selected non-zero entries as the perturbation matrix constitutes an even sparser matrix $\mathbf{\Delta A}$. This matrix is clearly non-negative and symmetric, therefore, a well-known upper bound for its largest eigenvalue can be derived from the Perron–Frobenius theorem and the definition of matrix norms:

$$\lambda_n^{\Delta A} \leq \max_{i \in \{1,\ldots,N\}} \sum_{j=1}^{N} (\Delta A)_{ij}.$$

This bound states that the largest eigenvalue is not larger than the maximal row-sum of the $\mathbf{A}$'s elements. A proof of this theorem can be found in Cvetkovic et al. (2009, Proposition 1.1.1). Recall that $\mathbf{\Delta A}$ is selected form the non-zero entries of $\mathbf{A}$, therefore, it can neither have more non-zero row-entries nor different ones. As a consequence the upper bound for its spectral norm cannot be larger. In fact, when p is small, $\mathbf{\Delta A}$ contains very few non-zero entries and it's upper bound is almost certainly significantly smaller. For example, consider Figure 3.1c which represents a fraction $p = 0.1$ of the edges in Figure 3.1d. Almost all matrix entries are 0. This is representative of $\mathbf{\Delta A}$'s expected relative sparsity.

Empirically, a relationship of $\|\mathbf{\Delta A}\| \approx O(p\|\mathbf{A}\|)$ is observed for the graphs used in this study. Results for $p = 0.1$ are presented in Table 3.1.

A proportional relationship for subgraphs sampled with random edge selection is also reported in Leskovec and Faloutsos (2006). Therefore, there is theoretical and empirical evidence supporting condition (1.) when the perturbation is small as defined in Lü et al. (2015).

It is more difficult to find clear support for the second condition. There is much evidence that the gaps between the largest adjacent eigenvalues are almost always significant (c.f., de Aguiar and Bar-Yam, 2005, Farkas et al., 2001). When these gaps are roughly preserved after perturbation condition (2.) can be supported for the leading eigenpairs. But this argument does not hold for the bulk of the eigenvalues in the middle of the spectrum. The same studies indicate their gaps are very small and that they distribute close to zero.

Table 3.1: *Spectral norm comparison between adjacency matrix and perturbation matrix*

Graph	N	$\|\mathbf{A}\|$	$\|\mathbf{\Delta A}\|$	$p\|\mathbf{A}\|$
florida	128	39.63	5.03 ± 0.18	3.96
jazz	198	40.03	5.20 ± 0.24	4.00
neural	297	24.37	4.24 ± 0.31	2.44
USAir	332	41.23	5.48 ± 0.28	4.12
netscience	379	10.38	2.56 ± 0.31	1.04
metabolic	453	26.58	5.32 ± 0.40	2.66
email	1133	20.75	3.66 ± 0.25	2.07
hamster	1788	46.16	6.37 ± 0.29	4.62
yeast	2224	19.49	3.76 ± 0.25	1.95

Values for $\|\mathbf{\Delta A}\|$ are reported in the format <mean> ± <standard deviation> for 30 independent randomly selected perturbation sets with $p = 0.1$.

Luckily, their eigenvector shifts are not very harmful. Consider the eigendecomposition in Equation (3). When the eigenvalues are almost zero, the eigenvectors interact only weakly with the adjacency matrix. Therefore, even large eigenvector shifts in this segment of the spectrum should have only a small effect.

However, there is little theoretical support for these statements. The classical theorem of Weyl (1912) only gives a very loose upper bound on the eigenvalue perturbation of $|\lambda_i^r - \lambda_i| \leq \|\mathbf{\Delta A}\|$. A recent study by Eldridge et al. (2017) provides a tighter upper bound for the eigenvalue perturbation of order $O(\sqrt{\log N})$ in matrices with block-constant structure (stochastic block models, similar to community structure observed in real-world data). The same work also gives a better bound for eigenvector perturbation and shows that is is small in such settings. Nevertheless the eigengaps are given by the distribution of the spectrum of $\mathbf{A^r}$ which is data-dependent, a circumstance not controlled by this method.

In summary, when using graphs representing real-world data and small perturbations the assumption of invariant eigenvectors can be valid and is supported by empirical evidence. The supplementary material of Lü et al. (2015, Figure S3) shows that the structural consistency remains stable for various values of p up to $p = 0.5$. This validates the stability of the approach on non-random matrices even if it can sound surprising that removing 50% of the edges still produces consistent results. At the same time it must not be forgotten that the distribution of the eigenvalues in the input data is not controlled and when it happens to be unsuitable due to special circumstances in the structure of $\mathbf{A^r}$, the eigenpairs may be sensitive to the perturbation.

3.2.3 Structural Consistency for Degenerate Spectra

The structural consistency theory described in Section 3.2 assumes all eigenvalues of $\mathbf{A^r}$ are simple. However, the spectrum will sometimes be degenerate. When an eigenvalue is simple, the corresponding eigen-subspace (eigenvector) is determined by a line in the eigenspace orthogonal to all eigen-subspaces corresponding to the different eigenvalues. When an eigenvalue has multiplicity larger than one then any vector in the corresponding eigen-subspace is orthogonal to all other eigen-subspaces and there are infinite possibilities

to choose a set of (not necessarily orthogonal) eigenvectors corresponding to this eigenvalue. The occurrence of eigenvalues with multiplicity larger than one can have structural or coincidental reasons. In any case, the introduced perturbation can disturb this degeneracy and therefore N distinct eigenvalue perturbations ($\Delta\lambda$) have to be obtained to perturb the eigenvalues which may then not be degenerate anymore. The approach taken by Lü et al. (2015) is described and explained in more detail below.

Let $\mathbf{A^r}$ have $M \leq N$ unique eigenvalues λ^r_m, each corresponding a set of $V_m \geq 1$ orthonormal eigenvectors u^r_{mv} where V_m is the multiplicity of λ^r_m. In total this defines N unperturbed eigenpairs $(\lambda^r_{mv}, u^r_{mv})$ such that

$$\lambda^r_{mv} = \lambda^r_m \qquad \forall m \in \{1, \ldots, M\} \quad \forall v \in \{1, \ldots, V_m\}.$$

As explained above, any linear combination of eigenvectors u^r_{mv} is an eigenvector corresponding to λ^r_m. Therefore,

$$\bar{u}_{mv} = \sum_{w \in V_m} \beta_{mw} u^r_{mw} \tag{40}$$

can be chosen as the eigenvector corresponding to λ^r_{mv} and the unperturbed eigenfunction becomes

$$\mathbf{A^r}\bar{u}_{mv} = \lambda^r_{mv}\bar{u}_{mv}. \tag{41}$$

Due to the assumption of invariant eigenvectors (see Section 3.2) $\bar{u}_{mv}$ is also the perturbed eigenvector corresponding to $(\lambda^r_{mv} + \Delta\bar{\lambda}_{mv})$. The perturbed eigenfunction thus becomes

$$(\mathbf{A^r} + \mathbf{\Delta A})\,\bar{u}_{mv} = \left(\lambda^r_{mv} + \Delta\bar{\lambda}_{mv}\right)\bar{u}_{mv}$$

which can be expanded and simplified by using (41):

$$\mathbf{\Delta A}\bar{u}_{mv} = \Delta\bar{\lambda}_{mv}\bar{u}_{mv}.$$

Replacing the the eigenvectors with (40) and left multiplying with $(u^r_{mi})^\top$ (consider the orthonormality of the original eigenvectors) gives an equation for each $i \in \{1, \ldots, V_m\}$:

$$\sum_{w \in V_m} (u^r_{mi})^\top \mathbf{\Delta A} u^r_{mw} \beta_{mw} = \Delta\bar{\lambda}_{mv}\beta_{mi}. \tag{42}$$

Equation (42) is an eigenfunction defined for each eigenvalue λ^r_{mv}:

$$\begin{pmatrix} (u^r_{m1})^\top \mathbf{\Delta A} u^r_{m1} & \ldots & (u^r_{m1})^\top \mathbf{\Delta A} u^r_{mV_m} \\ \vdots & \ddots & \vdots \\ (u^r_{mV_m})^\top \mathbf{\Delta A} u^r_{m1} & \ldots & (u^r_{mV_m})^\top \mathbf{\Delta A} u^r_{mV_m} \end{pmatrix} \begin{pmatrix} \beta_{m1} \\ \vdots \\ \beta_{mV_m} \end{pmatrix} = \Delta\bar{\lambda}_{mv} \begin{pmatrix} \beta_{m1} \\ \vdots \\ \beta_{mV_m} \end{pmatrix}$$

Writing the column vectors of β_{mw} as b_{mv} and the $V_m \times V_m$ matrix as $\mathbf{M}$ where

$$M_{iw} = (u^r_{mi})^\top \mathbf{\Delta A} u^r_{mw} \tag{43}$$

shows this in a more recognizable form:

$$\mathbf{M}b_{mv} = \Delta\bar{\lambda}_{mv}b_{mv}. \tag{44}$$

The eigendecomposition of $\mathbf{M}$ provides the orthonormal coefficient vectors b and the eigenvalue perturbations $\Delta\bar{\lambda}$. The chosen set of eigenvectors $\bar{u}$ can then be computed using (40).

The perturbed adjacency $\hat{\mathbf{A}}$ matrix is computed using Equation (39) from the non-degenerate case. For every degenerate eigenpair, an eigendecomposition of $\mathbf{M}$ is computed and the non-degenerate perturbation term $\Delta\lambda_i$ as well as the non-degenerate eigenvector u_i^r are replaced with $\Delta\bar{\lambda}_{mv}$ and $\bar{u}_{mv}$ respectively.

3.3 Perturbation Approach to Link Prediction (SPM)

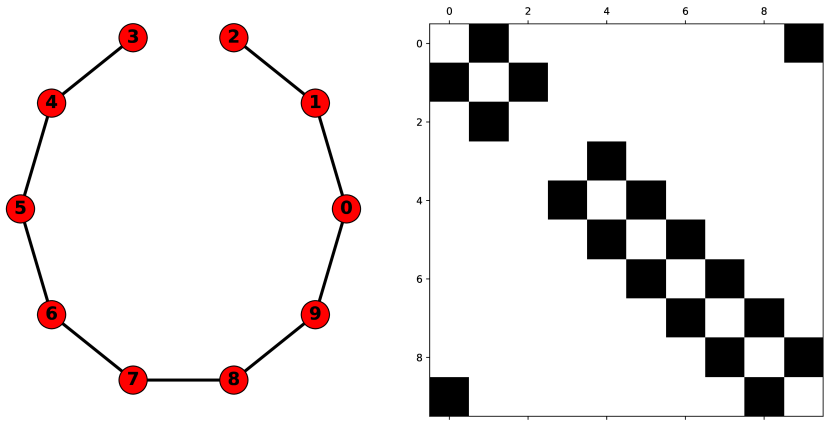
Lü et al. (2015) extended the structural consistency measure that is described in Section 3.2 and defined the Structural Perturbation Method (SPM) for link prediction. SPM aims to find a set of *missing edges* which are interpreted as a perturbation to the input graph. In a temporal sense, missing edges can be seen as edges that will exist in the future. This is the canonical application of link prediction; to forecast the evolution of edges in a graph. Equivalently it can be applied in a setting where the input graph is incomplete because some edges may be hidden due to missing information or because there is a less than one probability of observing existing edges. The question is: how can structural consistency be used to discover them? The obvious problem is of course that the missing edge set, i.e. the perturbation, is unknown.

Recall that the approximation of the perturbed adjacency matrix is $\hat{\mathbf{A}}$ (defined by Equation 39). It is interpreted as a score matrix and used to rank unobserved edges between the vertices of G^r. The unobserved edges with the largest values in $\hat{\mathbf{A}}$ are considered the most likely candidates to be perturbations. This process appears to be suitable for link prediction with the perturbation representing the missing edges. However, as stated before: neither the perturbation nor the perturbed graph are known. This is different from the structural consistency setting.

Let the perturbed graph $G(V, E)$ be the graph that is *observed* by some application. The observing application wants to find missing edges in relation to some unknown graph that represents either a future state or a better observation with recovered hidden edges. The perturbation set ΔE is selected from the observed graph by random selection of known edges. The unperturbed graph is defined as $G^r(V, E^r = E - \Delta E)$.

As an example, the link prediction scenario is applied to the graph and matrices in Figure 3.1. Let the ring graph in Figure 3.1a with *missing* edge $(2, 3)$ be the observed graph G. Figure 3.3a depicts this graph as it is observed by an application in the link prediction scenario. From this graph, a fraction p of the observed edges is randomly selected to define a perturbation set ΔE and an unperturbed graph G^r can be defined that has those edges removed. Notice that the perturbed graph in the previous scenario is an unknown graph not represented by either G or G^r. This unknown graph contains the *missing* edge $(2, 3)$ but this edge is not observed and also not chosen as the perturbation set. The link prediction algorithm is going to predict the unknown graph by predicting the *missing* edge.

To understand how SPM recovers missing edges, recall the assumption that the observed graph G has a structural pattern that determines its eigenpairs and remains stable



(a) *The observed graph* $G(V, E)$. **(b)** *The observed adjacency matrix* $\mathbf{A}$.

Fig. 3.3: *Link prediction scenario using the ring graph from Figure 3.1.*

under small perturbations. The assumption can be extrapolated to the unknown graph. The unknown graph is obtained by adding the missing links to G, which constitutes a small perturbation. Furthermore, any small and random perturbation to G^r should have a similar effect as this small perturbation to G since the structural properties do not change significantly under such perturbations.

Now let us translate the approach to the corresponding adjacency matrices. The application randomly generates small perturbations $\mathbf{\Delta A}$. Each random perturbation induces a new unperturbed matrix $\mathbf{A^r}$ that is perturbed using Equation (39) to obtain an approximation $\mathbf{\hat{A}}$ which should rank the perturbed edges high. Obviously the application already knows these edges. However, $\mathbf{\hat{A}}$ ranks not only perturbed edges but all the unobserved edges. Therefore, some edges that have not been seen before and are well explained by the eigenpairs of $\mathbf{A^r}$ will also be ranked higher than those that are not supported by the observed structure. An example of this process is depicted in Figure 3.4, which shows the result of multiple first-order approximations. While some perturbations have very little effect (right most), others (left and middle) rank more unobserved edges high than those in the perturbation set. Repeating this process many times reduces some of the random effects, and the edges that are frequently ranked high are considered candidate missing edges. These candidate edges are resilient to random perturbations. In Figure 3.4 (left and middle) the missing edge $(2, 3)$ is ranked high in both manifestations, even though it has neither been in the perturbation set nor observed. Conversely, when $\mathbf{\hat{A}}$ is found to be highly sensitive to the perturbation set, one would not expect significant differences among the unobserved edges. Such a behavior would suggest that no dominant pattern exists in G and then the graph is not well predictable using its eigenpairs.

In summary, a solution to SPM is computed by performing $s \in \mathbb{Z}^+$ independent perturbations. Each perturbation $i \in \{1, \ldots, s\}$ is defined by randomly selecting a fraction p of the edges from the observed graph. Then the perturbed matrix approximation $\mathbf{\hat{A}_i}$ is

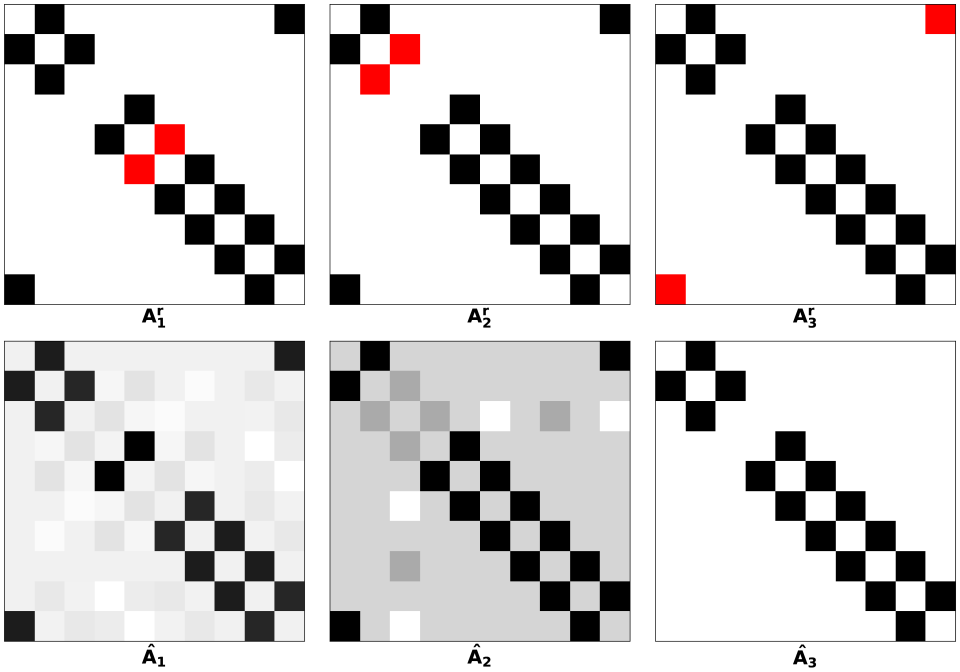


Fig. 3.4: *Three independent random perturbations of the observed adjacency matrix. The matrix depicted in Figure 3.3b is the perturbed matrix. In the top row, each matrix is a different unperturbed matrix* $\mathbf{A}_\mathbf{i}^\mathbf{r}$ *(black cells) with the corresponding perturbation* $\mathbf{\Delta A}$ *(red cells). In the bottom row, each matrix is the first-order approximation* $\hat{\mathbf{A}}_\mathbf{i}$ *corresponding to the perturbation of the matrix above.*

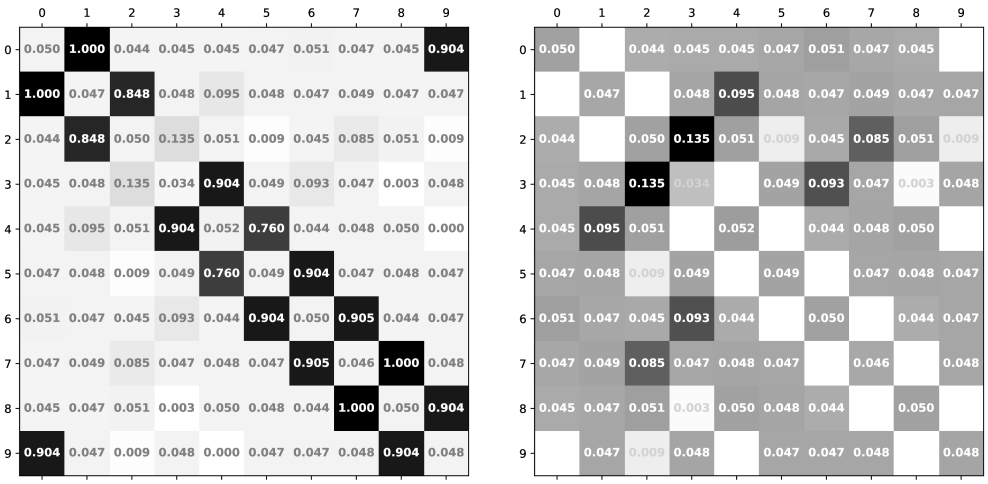
defined equally as $\hat{\mathbf{A}}$ for structural consistency. Finally all approximations are averaged:

$$\langle \hat{\mathbf{A}} \rangle = \frac{1}{s} \sum_{i=1}^{s} \hat{\mathbf{A}}_\mathbf{i} \tag{45}$$

The solution to the link prediction problem is given by choosing the $p*|E|$ top-ranked edges in $\langle \hat{\mathbf{A}} \rangle$ as the missing edges. The SPM link prediction process is more formally defined by Algorithm 3.2. An example manifestation of $\langle \hat{\mathbf{A}} \rangle$ for $s = 10$ is shown in Figure 3.5. Each element in $\langle \hat{\mathbf{A}} \rangle$ can be interpreted as an estimate for the existence of an edge. Therefore, the score matrix is also called the *link-existence estimator*. This method is fully evaluated and compared to state-of-the-art link prediction algorithms in Lü et al. (2015).

3.3.1 Related Work

The landscape of link prediction methods on complex graphs is versatile but most approaches can be said to compute a measure of similarity between vertices to identify edges between them. In Section 3.1, the distinction between local and global methods is introduced and for detailed information about the state-of-the-art the reader is referred to surveys published in Liben-Nowell and Kleinberg (2007), Lü and Zhou (2011) and Wang



(a) *A link-existence estimator* $\langle \hat{\mathbf{A}} \rangle$. **(b)** *Heatmap of unobserved edges for* $\langle \hat{\mathbf{A}} \rangle$.

Fig. 3.5: *An example for a link-prediction result. (a) A manifestation of* $\langle \hat{\mathbf{A}} \rangle$*. The corresponding observed graph is given in Figure 3.3a and* $s = 10$ *independent perturbations have been averaged. (b) The corresponding heatmap of unobserved edges. Clearly, the missing edge* $(2, 3)$ *has been recovered.*

.

and Liao (2014). These articles cover most local and quasi-local methods and random walks techniques. For the purposes of contextualizing the SPM method, two of these approaches are described below.

As mentioned in the chapter introduction, SPM is a global method. It takes the entire graph as input and computes scores for all possible edges based on characteristics present in a graph matrix. One of the earliest global approaches that used the full adjacency matrix to compute similarity scores for all vertices is the Katz index (Katz, 1953). It exploits that the elements of $\mathbf{A}^k$, that is, the adjacency matrix to the power of k, contain the number of paths of length k between corresponding vertices. This can be generalized to a count of paths up to k with exponential damping, such that shorter paths are more important for vertex similarity.

Another class of link prediction methods are maximum likelihood methods. A strong assumption about the graph structure can often be made in advance and expressed as a graph model. This enables the generation of model instances with parameters set to be similar to an observed graph. Instances can be interpreted as ideal versions of the observed graph. The difference between the observed and the ideal graph is evidence for missing or erroneous edges. A general framework employing this approach is using stochastic block models (SBM, Guimerà and Sales-Pardo, 2009) which model community structure in graphs. Basically, many SBM models are generated and compared to the observed graph by maximum likelihood analysis. This allows to assign probabilities to unobserved edges based on whether unobserved links exist in similar SBM instances.

Shortly after the publishing the SPM method, another global link prediction method was presented by Pan et al. (2016). Based on similar ideas as the stochastic block model the authors propose a maximum likelihood analysis approach based on a graph model defined by powers of the adjacency matrix that represent the number of loops existing between vertices. It generalizes the observation that triangles, that is loops of length three, play an important role in explaining the higher-order organization of graphs. The number of loops of length k is related to the leading eigenvalues of the adjacency matrix $\mathbf{A}^k$; an idea very similar to the Katz index. Pan et al. show that their loop-based method is very accurate in identifying missing and spurious edges. However, its calculation is computationally more intensive than SPM.

Pech et al. (2017) confirm the high prediction accuracy of the loop method and also that of SPM in an extensive evaluation. A new method based on robust principal component analysis (robust PCA) is also introduced. It is more versatile and computationally efficient than SPM or the loop method but less accurate. PCA is used to extract a "backbone" structure of a graph which is taken to represent the "true graph" while the remaining structure is interpreted as noise or errors.

The robust PCA approach is combined with SPM by Xu et al. (2017). SPM is used for link prediction and robust PCA is applied to the result of SPM to remove noise added by the perturbations. This is an interesting approach as it does not reduce the computational complexity at all. Another important contribution in this article is the extension of SPM to directed graphs by perturbing only the symmetric part of the adjacency matrix.

Unrelated to robust PCA, Wang et al. (2017) propose a popularity based SPM algorithm for link prediction in evolving graphs (PBSPM). It applies SPM to weighted matrices that represent the time-varying mutual attractiveness of vertices in a graph. The authors contribute a "fast" SPM algorithm which is essentially a truncated SPM that ignores all but very few leading eigenpairs. This approach is demonstrated to be more accurate than SPM in a limited evaluation on four graphs that model time-evolving popularity-based interactions.

Recently, Zeng et al. (2018c) evaluated SPM in bioinformatics for the prediction of disease-associated micro RNA structures. A case study confirms that SPM works robustly and accurately in recovering potentially breast-cancer related structures that were previously selected by experts.

3.4 Computational Aspects of SPM

This section presents and analyzes topics related to the computation of SPM link prediction. The discussion is based on the definitions in Sections 3.2 and 3.3. While the procedure is loosely described in Lü et al. (2015), the authors did not publish any source code for SPM and therefore the algorithms presented below have been re-implemented independently. They may differ from the original work regarding implementation details that were not described. Afterwards, the computational cost in terms of floating point operations is analyzed. The section concludes with a time complexity analysis of SPM.

3.4.1 Algorithms

Here, the algorithms implemented for SPM link prediction are presented. Appendix A lists the notation conventions and defines the functions.

Algorithm 3.1 details the process for the calculation of the first-order approximation $\hat{\mathbf{A}}$ of the perturbed matrix $\mathbf{A}$. This is implemented according to the method described in Section 3.2. The algorithm takes as input the unperturbed matrix and the perturbation matrix. This is the core step of calculating the structural consistency σ_c of a graph and used to rank edges and very important in the SPM link prediction method to calculate the link existence estimator.

Algorithm 3.1 Approximation of $\hat{\mathbf{A}}$

Input: $\mathbf{A^r}, \mathbf{\Delta A}, N$

$\Lambda^r, \mathbf{U^r} \leftarrow \textsc{eigh}(A^r)$
$\hat{\mathbf{A}} \leftarrow \textsc{zeros}(N, N)$
$n \leftarrow 1$
for all $(\lambda, V_m) \in \textsc{unique}(\Lambda^r)$ **do**
  if $V_m > 1$ **then** ▷ *Degenerate Eigenproblem, see Sec. 3.2.3*
    $\mathbf{M} \leftarrow \textsc{empty}(V_m, V_m)$
    for all $i \in V_m$ ***do***
      $u_i \leftarrow \mathbf{U^r}(n + i - 1)$
      for all $w \in V_m$ ***do***
        $u_w \leftarrow \mathbf{U^r}(n + w - 1)$
        $\mathbf{M}(i, w) \leftarrow u_w^\top \mathbf{\Delta A} u_i$ ▷ *Eq.* (43)
    $\Delta\bar{\Lambda}, \mathbf{B} \leftarrow \textsc{eigh}(\mathbf{M})$ ▷ *Eq.* (44)
    for all $v \in V_m$ ***do***
      $\Delta\bar{\lambda} \leftarrow \Delta\bar{\Lambda}(v)$
      $\bar{u} \leftarrow \sum_{j \in V_m} \mathbf{B}(j, v)\mathbf{U^r}(n + j - 1)$ ▷ *Eq.* (40)
      $\hat{\mathbf{A}} \leftarrow \hat{\mathbf{A}} + (\lambda + \Delta\bar{\lambda})\bar{u}\bar{u}^\top$
  else
    $u \leftarrow \mathbf{U^r}(n)$
    $\Delta\lambda \leftarrow u^\top \mathbf{\Delta A} u$ ▷ *Eq.* (38)
    $\hat{\mathbf{A}} \leftarrow \hat{\mathbf{A}} + (\lambda + \Delta\lambda) u u^\top$ ▷ *Eq.* (39)
  $n \leftarrow n + V_m$

Output: $\hat{\mathbf{A}}$

Algorithm 3.2 summarizes the calculation of the link existence estimator $\langle \hat{\mathbf{A}} \rangle$ as used in SPM (see Section 3.3). It takes as input the input graph G and the parameters p and s controlling the fraction of edges to perturb and the number of perturbations to average over respectively.

Parallelization of the Algorithms: In the scope of this work no attempt has been made to optimize above algorithms for any specific execution environment. But it is easy to see that above algorithms can be naively parallelized without major change.

Each of the iterations of Algorithm 3.2 is completely independent and can be computed in parallel. Note however that this requires significant memory workspace when $\mathbf{A}$ is large. $\mathbf{A}$ and $\langle \hat{\mathbf{A}} \rangle$ can be stored in shared memory. Additionally, space to store s

Algorithm 3.2 Calculation of Link Existence Estimator $\langle \hat{\mathbf{A}} \rangle$

Input: $G(V, E), p, s$

1: $N \leftarrow |V|$
2: $\langle \hat{\mathbf{A}} \rangle \leftarrow \text{ZEROS}(N, N)$
3: $e_p \leftarrow \lfloor p * |E| \rfloor$ ▷ *Fraction p of number of edges, rounded down to closest integer*
4: ***for*** $i = 1; i <= s; i \leftarrow i + 1$ ***do***
5: $\quad \Delta E \leftarrow \text{RANDOMCHOICE}(E, e_p)$ ▷ *Indices of the edges chosen as perturbation set*
6: $\quad E^r \leftarrow E - \Delta E$
7: $\quad G^r \leftarrow (V, E^r)$
8: $\quad \mathbf{A^r} \leftarrow \text{A}(G^r)$
9: $\quad \mathbf{\Delta A} \leftarrow \text{A}(G) - \mathbf{A^r}$
10: $\quad \hat{\mathbf{A}} \leftarrow \text{PERTURB}(\mathbf{A^r}, \mathbf{\Delta A}, N)$ ▷ *Algorithm 3.1*
11: $\quad \langle \hat{\mathbf{A}} \rangle \leftarrow \langle \hat{\mathbf{A}} \rangle + \hat{\mathbf{A}}$
12: ***return*** $\frac{1}{s} * \langle \hat{\mathbf{A}} \rangle$

versions of $\mathbf{A^r}$ and $\hat{\mathbf{A}}$ in process memory and the workspace required for s concurrent eigendecompositions (Line 1) needs to be considered. This is less of a problem in a distributed system where each iteration is computed in its own workspace. It is possible to parallelize Algorithm 3.1 in the same manner. $\mathbf{\Delta A}$ and the eigenpairs can be stored in shared memory. However, N copies of an $N \times N$ matrix are then produced in Line 16. which is infeasible. This issue can be mitigated by using a process pool to run only a small number of iterations concurrently. An even more efficient parallelized algorithm is possible by computing the perturbed matrix element- or blockwise but this not as straight forward.

3.4.2 Operation Count

This section presents an analysis of the computational cost for the SPM link prediction method of Lü et al. (2015). The cost is quantified in terms of floating point operations (flop) required to compute a result. This estimation is based on one flop representing a single addition, subtraction, multiplication, or division which leads to the following definitions for the cost of higher order operations on matrices:

- The scalar product of two vectors with N elements requires $2N$ flop.
- The matrix product of a $N \times N$ matrix with a $N \times 1$ matrix (column vector) requires $2N^2$ flop.
- The matrix product of a $M \times N$ matrix and a $N \times L$ matrix requires $2MNL$ flop.

These definitions are used to estimate the operation count for dense matrices, i.e matrices with relatively few non-zero elements. Because this is not always the case, estimation is then extended to consider matrix sparsity. Furthermore, it is assumed that standard numerical solvers, as those implemented in LAPACK[1] or MATLAB[2], are used.

[1] http://www.netlib.org/lapack/
[2] https://www.mathworks.com/products/matlab.html

Computation Steps: When all eigenvalues are simple, each SPM iteration consists of the following steps that incur significant computational cost. Operations before Line 10 in Algorithm 3.2 are considered data preparation (which can be pre-computed) or computationally insignificant and are therefore ignored. Their cost does not affect the complexity analysis below and it would create an unwanted dependency on implementation specific details.

1. (Alg. 3.1, Line 1) A single spectral decomposition is computed to obtain all eigenpairs of matrix $\mathbf{A^r}$. The operation count depends on properties of the matrix and the exact algorithm that is executed in the numerical computation library. Anderson et al. (1999, Table 3.13) report that LAPACK driver routines for generic square matrices require $26.22N^3$ operations. Demmel et al. (2008) report that a fast algorithm on tridiagonal matrices rquires $O(N^{2.8})$ flop. However, the reduction of a symmetric matrix to tridiagonal form costs $\frac{8}{3}N^3$ flop (Demmel, 1997, p. 211). Since there are no definitive numbers, ηN^3 flop will be used, where η is a performance constant depending on the eigensolver and $0 < \eta \ll N$.
2. (Alg. 3.1, Line 19) The approximation of the eigenvalue perturbation (Equation 38) requires one matrix-vector product ($2N^2$ flop) and one vector inner product ($2N$ flop). That is a total of $2N^2 + 2N$ flop *per eigenvalue*.
3. (Alg. 3.1, Line 20) The update to the perturbed matrix $\hat{\mathbf{A}}$ (see Equation 39) requires a vector outer product equivalent to a matrix product of a $N \times 1$ and a $1 \times N$ matrix ($2N^2$ flop), one floating-point addition (N flop), and one addition of two $N \times N$ matrices corresponding to (N^2 flop). This is a total cost of $3N^2 + N$ flop *per eigenvalue*.
4. (Alg. 3.2, Line 11) The update of the link existence estimator $\langle \hat{\mathbf{A}} \rangle$ requires another N^2 flop per independent perturbation.
5. (Alg. 3.2, Line 12) N^2 flop are required to calculate the element-wise average of the link existence estimator $\langle \hat{\mathbf{A}} \rangle$.

Steps 1-4 are executed s times as SPM performs multiple perturbations. Let M be the number of eigenvalues; the method has an estimated cost of

$$s\left(\eta N^3 + 5MN^2 + 3MN + N^2\right) + N^2 \tag{46}$$

floating point operations. For a non-degenerate spectrum, i.e. $N = M$, the cost is $s\left(6\eta N^3 + 4N^2\right) + N^2$ flop.

Degenerate Spectra: Let V_m denote the multiplicity of eigenvalue λ_m^r. When $M < N$, i.e. some eigenvalues have multiplicity $V_m > 1$, Algorithm 3.1 uses the degenerate eigenproblem branch which replaces Steps 2 and 3 for each such eigenvalue with:

- (Alg. 3.1, Line 11) $V_m(2N^2 + 2V_m N)$ flop for the creation of $\mathbf{M}$.
- (Alg. 3.1, Line 12) Approximately ηV_m^3 flop for an additional eigendecomposition of the dense matrix $\mathbf{M}$.
- (Alg. 3.1, Line 15) $V_m^2 N$ flop for the computation of the eigenvectors $\bar{u}$.
- (Alg. 3.1, Line 16) $V_m(3N^2 + N)$ for the update to perturbed matrix $\hat{\mathbf{A}}$.

In summary, the estimated operation count for degenerate spectra is

$$s\left(\eta N^3 + 5MN^2 3V_m + 3V_m^2 MN + MNV_m + \eta V_m^3 M + N^2\right) + N^2 \tag{47}$$

floating point operations; assuming all V_m are approximately equal. Therefore, large eigenvalue multiplicities will increase the cost of SPM as the degenerate branch of Algorithm 3.1 is clearly more expensive, especially if for some reason the largest multiplicity is of $O(N)$.

In this work SPM is applied to graphs derived from real-world observations. For such graphs the bulk of the eigenvalues should distribute close to zero (Preciado and Rahimian, 2017) and some real-world data derived graphs exhibit large zero-multiplicities (see de Aguiar and Bar-Yam, 2005, Farkas et al., 2001). However, in this work most zero eigenvalues are eliminated due to the contributions in Chapters 5 and 6 that tend to eliminate and disturb automorphisms. Therefore, there is little reason to expect any V_m to be large. Eigenvalues with exactly equal numerical value occur very rarely in the evaluations conducted for this dissertation. For $M \approx N$ the difference between (46) and (47) is not meaningful. Therefore, a special consideration of the degenerate case is forgone in the remainder of this work. Nevertheless spectral degeneracy must be reconsidered when choosing to apply SPM in different contexts.

Sparse Matrix Operations This work makes the assumption that sparse matrix operations have a cost proportional to the number of non-zero matrix elements (nnz) and it defines sparse as $nnz = O(N)$. All dense matrices in this section have $nnz = N^2$. The matrices $\mathbf{U}^r$ (eigenvectors) and $\hat{\mathbf{A}}$ (perturbed approximation) are dense matrices. $\mathbf{A^r}$ is assumed to be a sparse matrix, and therefore $\mathbf{\Delta A}$ is sparse as well. This has the following consequences:

- The sparsity of $\mathbf{A^r}$ affects Step 1. However, the effect is difficult to quantify. To the best of my knowledge, the standard numerical solvers compute all eigenpairs of a sparse symmetric matrix using the same algorithms as on dense matrices. This concerns primarily the tridiagonalization routine. Therefore this work assumes the same cost of ηN^3 flop.
- Step 2 is clearly cheaper due to sparsity of $\mathbf{\Delta A}$. When $\mathbf{A^r}$ is sparse, it has $nnz = cN$ for a constant $0 < c \ll N$. Then by definition $\mathbf{\Delta A}$ has $nnz = pcN$ for a constant $0 < p \ll 1$ and Step 2 is discounted proportionally by a factor $(pcN)/N^2$. In total the cost becomes $pc(2N + 2)$ flop *per eigenvalue*.

The estimated cost for a sparse non-degenerate input matrix is

$$s\left[(3+\eta)N^3 + (2+2pc)N^2 + 2pcN\right] + N^2 \tag{48}$$

floating point operations with p a parameter of SPM for the fraction of links to perturb and $c = (1-p)|E|/|V|$ is given by the input graph $G(V, E)$.

Summary: The computational cost of SPM when $\mathbf{A}^{\mathbf{r}}$ is sparse and has nearly non-degenerate spectrum has been analyzed above. Ignoring lower order terms in Equation (48) gives an estimate of

$$(3+\eta)sN^3 + O(N^2) \tag{49}$$

floating point operations. The constant η is determined by the performance of the eigensolver routine and s is the SPM parameter for the number of iterations of the algorithm.

3.4.3 Time Complexity Analysis

The operation count analysis in Section 3.4.2 allows to estimate the time complexity of SPM. Assuming all floating point operations take equal time and are executed in series, Equation (49) gives an estimated time complexity of SPM. However, these assumptions are not realistic. A multitude of issues related to computer hardware architecture influence how long the execution of an algorithm takes. There are also probabilistic elements in algorithms that affect runtime. Finally, some operations are executed in parallel and not in series. However, the purpose of this analysis is not to predict algorithm runtime but to determine the order of complexity of SPM. This complexity is then compared with that of the derived methods CGSPM in Chapter 5 and ECSPM in Chapter 6.

As established by Equation (49), the complexity is $O(N^3)$. From the computational cost analysis we know that the main contributors are the eigendecomposition with eN^3 flops per iteration and the computation of the perturbed adjacency matrix with $3N^3$ flops per iteration.

Concerning the eigendecomposition of matrix $\mathbf{A}^{\mathbf{r}}$, recall that standard solvers like MATLAB's *eig* or numpy's *eigh* use direct methods to solve symmetric eigenproblems for all eigenvalues and corresponding eigenvectors regardless of the sparsity of the matrix. The fastest method for large matrices currently used is a divide-and-conquer algorithm with time complexity $O(N^{2.5})$ (Demmel et al., 2008), which is lower than the measured number of operations $O(N^{2.8})$. However, all these methods have a complexity of $O(N^3)$ because they require an initial tridiagonalization costing $\frac{8}{3}N^3$ flop (see Demmel (1997), p. 211). It should be noted that iterative methods such as the the Lanczos iteration (see, Calvetti et al., 1994, Lanczos, 1950) can greatly exploit sparsity to reduce the operation count (refer to Section 2.4 for more details). According to Trefethen and Bau III (1997), one problem with iterative methods is that they are not always reliable for the computation of a full spectral decomposition because they might not converge on eigenvalues that are not well separated. In Equation (49) the spectral decomposition contributes a factor η. Assuming that the divide-and-conquer algorithm runs in time less than $O(N^3)$ means that the tridiagonalization routine is the limiting factor. Therefore, the performance constant $\eta \approx \frac{8}{3} = 2.\overline{6}$. Unless better eigensolvers are developed, the constant cannot be significantly improved.

The computation of the perturbed matrix has equal complexity of $O(N^3)$ and contributes a factor 3, as seen in Equation (49)). Unlike the eigendecomposition, the speed of the computation can be increased by parallelizing the Algorithms 3.1 and 3.2, as briefly discussed in Section 3.4.1.

In summary, the time complexity of SPM is $O(N^3)$, which is determined by the time complexities of the full eigendecomposition and the perturbed matrix calculation in about equal measure. While the former cannot be improved trivially, the latter can be easily improved by parallelization. Therefore, the eigendecomposition is the main bottleneck.

3.5 Conclusion

This chapter developed the theoretical foundations of SPM link prediction, which is based on eigenpair perturbation theory. It is an example of a complex computation problem that requires a full eigendecomposition of a graph matrix and has powerful applications in social network analysis as well as in information retrieval. It will be used to showcase the overarching hypotheses that such eigenvalue problems on graph matrices can be approximated accurately with lower complexity. The necessary extensions to achieve this will be developed in Chapters 5 and 6.

This chapter makes new contributions by providing a stability analysis and a computational cost analysis. The stability analysis investigates which graph properties are required for SPM to work. Specifically, it investigates the circumstance for eigenvectors of the adjacency matrix to remain stable under perturbation and concludes that SPM can be stable if the spectral distribution of the input graph shows well separated eigenvectors on the edges of the spectrum. Choosing unsuitable graphs may lead to eigenvectors that are very sensitive to perturbations and therefore low prediction accuracy or failure to make useful predictions.

The computational cost analysis establishes an estimate of the floating point operation count executed by SPM in terms of input graph size. This is used for a time complexity analysis that verifies the assumption that SPM has cubic time complexity and that this is primarily due to the complexity of the full eigendecomposition. This verifies part of the main hypothesis in this work and shows the need for more efficient approaches to solve eigenvalue problems on graphs. The computational cost and complexity analysis will be used to validate the above mentioned extensions in the following chapters.

Chapter 4

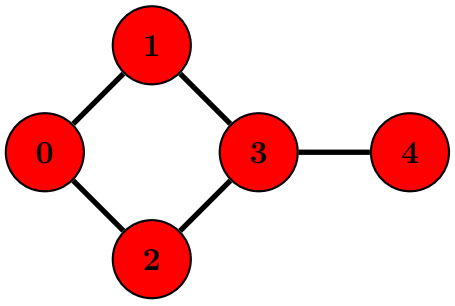
Spectral Coarse-Graining (SCG)

This chapter introduces spectral coarse-graining, a method to shrink adjacency matrices of graphs while preserving their eigenpairs, and discusses graph partitioning approaches needed for accurate coarse-graining.

4.1 Introduction

The Spectral Coarse-Graining Framework (SCG) published by de Lachapelle et al. (2008) shrinks matrices while attempting to maximally preserve a subset of eigenpairs. This chapter introduces the mathematical framework defining SCG for real and symmetric adjacency matrices and discusses the factors affecting its accuracy in detail. These topics will be relevant for the remainder of this work. The interested reader will find a more extensive treatment of this topic, proofs, and an extension to complex matrices in the original work (de Lachapelle et al., 2008).

Coarse-graining describes a process that removes detail from a graph. First, it computes a partitioning of all graph vertices into groups of similar vertices. Second, it constructs a coarse-grained graph that represents only relations between partitions. Mathematically, the graph adjacency matrix, which represents relationships between N atomic entities, is projected onto a subspace that represents only the relationships between k atomic partitions. This projection is reversible, although the information loss about the interior of each partition cannot be recovered. SCG is a coarse-graining method that optimizes for the maximal preservation of a chosen set of eigenpairs.



(a) *A small graph.*

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

(b) *The adjacency matrix of (a).*

Fig. 4.1: *An example of indistinguishable interactions. (a) Vertices* 1 *and* 2 *have the same set of neighbors* $\{0, 3\}$*. (b) Notice that the corresponding rows (red) in the adjacency matrix are equal. The contribution of duplicated rows to any linear transformation defined by the adjacency matrix is indistinguishable. Due to the symmetry of* $\mathbf{A}$*, the same is true for the corresponding columns.*

Vertices that are assigned to the same partition have to be as similar as possible to maximally preserve information in a coarse-graining. SCG defines vertex similarity as a generalization of the following equality. When two vertices have the same neighbors, their matrix elements in the corresponding rows (and columns) of the adjacency matrix are equal. Figure 4.1 presents an example. Assume the perspective of a random walker starting in Vertex 0. It can choose to go to either Vertex 1 or 2 but it does not change the probability of arriving at Vertex 3 and 4 in the following steps. The interactions of Vertex 1 are said to be *indistinguishable* from those of Vertex 2. Therefore, vertices with indistinguishable interactions and their corresponding adjacency matrix elements can be merged and represented as a single vertex without loss of information about their interactions. A simple way to do this in this particular case is to sum duplicated rows and columns in the adjacency matrix.

Recall that eigenpairs and vertices are related as defined in Section 2.3. Indistinguishable interactions occur when the eigenvector components associated to two given vertices v_l and v_m have the same value in every eigenvector. [3] Thus, indistinguishable vertices are formalized by the following condition.

$$\forall k \in \{1, \ldots, N\} : u_k(l) = u_k(m). \tag{50}$$

Unfortunately, duplicated rows are a special instance of linear dependency that does not occur frequently enough to allow for significant coarse-graining in real data. Therefore, the condition of indistinguishable vertices is relaxed to a more general relationship of vertex similarity defined as closeness of vertices in the eigenspace of the adjacency matrix.

The eigenspace coordinates of two vertices v_l and v_m are proportional to the set of their corresponding eigenvector components (see Section 2.3.4). Suppose their coordinates are almost equal as in (50). Using the eigendecomposition (3) for a single element of $\mathbf{A}$, their relationship can be expressed for $j \in \{1, \ldots, N\}$ as

$$\underbrace{\sum_{k=1}^{N} \lambda_k u_k(l) u_k(j)^\top}_{A_{lj}} + \underbrace{\sum_{k=1}^{N} \epsilon_k \lambda_k u_k(j)^\top}_{\text{error}} = A_{mj}$$

with each ϵ_k a small positive or negative real constant. Evidently, the error tends to zero as each ϵ_k becomes smaller. Therefore, if v_l and v_m have similar coordinates in the eigenspace, the rows in $\mathbf{A}$ should be similar as well. The inverse of this statement is not true however. Similar elements in $\mathbf{A}$ do not imply similar coordinates in the eigenspace.

To summarize, spectral coarse-graining uses the eigenvectors of $\mathbf{A}$ to assess vertex similarity. Similar vertices describe similar interactions in the adjacency matrix and merging them does not change the interactions described by the graph significantly.

The remainder of this chapter contains a detailed explanation of SCG to facilitate various definitions and discussions in Chapters 5 and 6. The content of this chapter is based primarily on the original work of de Lachapelle et al. (2008). Many aspects are

[3] Refer to Appendix E.1 for a proof.

extended with additional detail and improved error bounds for SCG with real matrices are derived. The remainder of this chapter is structured as follows. Section 4.2 defines the mathematical framework of SCG and the definition of a minimizing SCG projector, then, various approximate partitioning methods that can be used to obtain projectors are introduced and explained in Section 4.3. Finally, the chapter is concluded in Section 4.4 with a summary of aspects that are important for the contributions developed later in this dissertation.

4.1.1 Related Work

Initial work on Spectral Coarse-Graining for stochastic matrices was published in Gfeller and De Los Rios (2007) and the PhD thesis of Gfeller (2007) extended this work. The final framework was published in de Lachapelle et al. (2008). In Gfeller and De Los Rios (2008) an application of SCG to oscillator networks and its effect on network synchronization has been presented. Network synchronization is related to the spectrum of the Laplacian matrix of graphs.

Subsequent work using SCG has been focused on the area of network synchronization as well. Chen et al. (2013) investigated how SCG affects the synchronization in complex clustered graphs and find that a clear cluster-structure of the graph is important to preserve the relevant Laplacian eigenvalues, specifically, the first non-zero as well as the largest eigenvalue, both of which determine a network's synchronization ability. The influence of path length and degree distribution on network synchronization has been studied in Zeng et al. (2018a). Zeng et al. (2018b) present an approach that is almost equal to the k-means++ initialization method described in Section 4.3 of this work. Their method is introduced as a "new Spectral Coarse Graining algorithm" based on K-Means clustering (KCSCG). The empirical analysis on Laplacian matrices confirms the viability of this partitioning method for preserving network synchronization. However, De Lachapelle et al. (2008) have described this method in Section 5.4.2. (Method 2) of their publication ten years earlier.

4.2 The Spectral Coarse-Graining Framework

The following theory defines spectral coarse-graining in terms of a projection of the original graph's adjacency matrix onto a vector space where vertices with similar interactions become indistinguishable. Then, the same interactions can be represented with a smaller matrix.

Let $G(V, E)$ be a graph and $\mathbf{A}$ its real and symmetric adjacency matrix as defined in Section 2.2. In the SCG framework, $\mathbf{A}$ is referred to as the *original* matrix or system. Let $\mathbf{P} \in \mathbb{R}^{N\times N}$ denote a projection matrix and $\mathbf{L}, \mathbf{R} \in \mathbb{R}^{k\times N}$ with $k \leq N$ are called *semi-projectors*. They are defined by

$$\mathbf{P} \coloneqq \mathbf{R}^\top \mathbf{L}$$

such that

$$\mathbf{L}\mathbf{R}^\top = \mathbf{I}_k.$$

The semi projectors $\mathbf{L}$ and $\mathbf{R}$ are distinguished to conform with the definition in de Lachapelle et al. (2008) but due to the symmetry of the adjacency matrix in this work both semi-projectors are equal and $\mathbf{P}$ is an orthogonal projector of rank k. It follows that $\mathbf{P} = \mathbf{P}^\top$.

Given a graph adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, its coarse-graining $\tilde{\mathbf{A}} \in \mathbb{R}^{k\times k}$ is defined as

$$\tilde{\mathbf{A}} \coloneqq \mathbf{L}\mathbf{A}\mathbf{R}^\top \tag{51}$$

and the *reverse* projection of a coarse-graining back to a system of original size is

$$\mathbf{P}\mathbf{A}\mathbf{P} = \mathbf{R}^\top \tilde{\mathbf{A}}\mathbf{L}. \tag{52}$$

$\mathbf{P}\mathbf{A}\mathbf{P}$ is called the *projected system*.

There is the following relationship between the eigenpairs of $\mathbf{P}\mathbf{A}\mathbf{P}$ and $\mathbf{L}\mathbf{A}\mathbf{R}^\top$. For every non-zero eigenpair (μ, w) of the projection $\mathbf{P}\mathbf{A}\mathbf{P}$, $(\mu, \mathbf{L}w)$ is an eigenpair of $\mathbf{L}\mathbf{A}\mathbf{R}^\top$ and for every eigenpair $(\tilde{\lambda}, \tilde{u})$ of the coarse-grained system $\mathbf{L}\mathbf{A}\mathbf{R}^\top$, $(\tilde{\lambda}, \mathbf{R}^\top\tilde{u})$ is an eigenpair of $\mathbf{P}\mathbf{A}\mathbf{P}$. Therefore, there is a one-to-one mapping between the eigenpairs of the coarse-grained adjacency matrix and the projection $\mathbf{P}\mathbf{A}\mathbf{P}$.

Figure 4.2 presents a summary of the relationships between the different elements and operations in the SCG framework. In general, there is no transition from either the projected system or the coarse-grained system to the original system. That means, the original matrix can generally not be recovered because information is lost in the process. There is also no trivial projection of the eigenvalues from the original system to any of the other systems.

4.2.1 The Minimizing Projector

This section defines an optimization problem to find the optimal projector $\mathbf{P}$ and determines the minimizing projector that preserves a single eigenpair for a given graph partitioning. Section 4.3.4 extends this definition to the minimizing projector for multiple eigenpairs.

Let $\Gamma = \{\alpha_1, \ldots, \alpha_k\}$ denote a *partitioning* of the vertices in V into k non-empty *partitions* such that every vertex v is assigned to exactly one partition α. Suppose $\gamma : \{1, \ldots, N\} \to \{1, \ldots, k\}$ is a function that translates a vertex-index to its partition-index such that

$$\gamma(i) = g \iff v_i \in \alpha_g$$

and the multivalued function $\nu : \Gamma \to \{1, \ldots, N\}$ returns the indexes of all vertices in a given group:

$$\nu(\alpha) = \{i : \forall v_i \in \alpha\}.$$

A partition α is a set of vertices, but equivalently, it can be a set of eigenvector components that are associated to the vertices. This relationship has been defined in Section 2.3. The partitions are also called *groups* and the partitioning Γ is sometimes called *grouping* in the context of SCG.

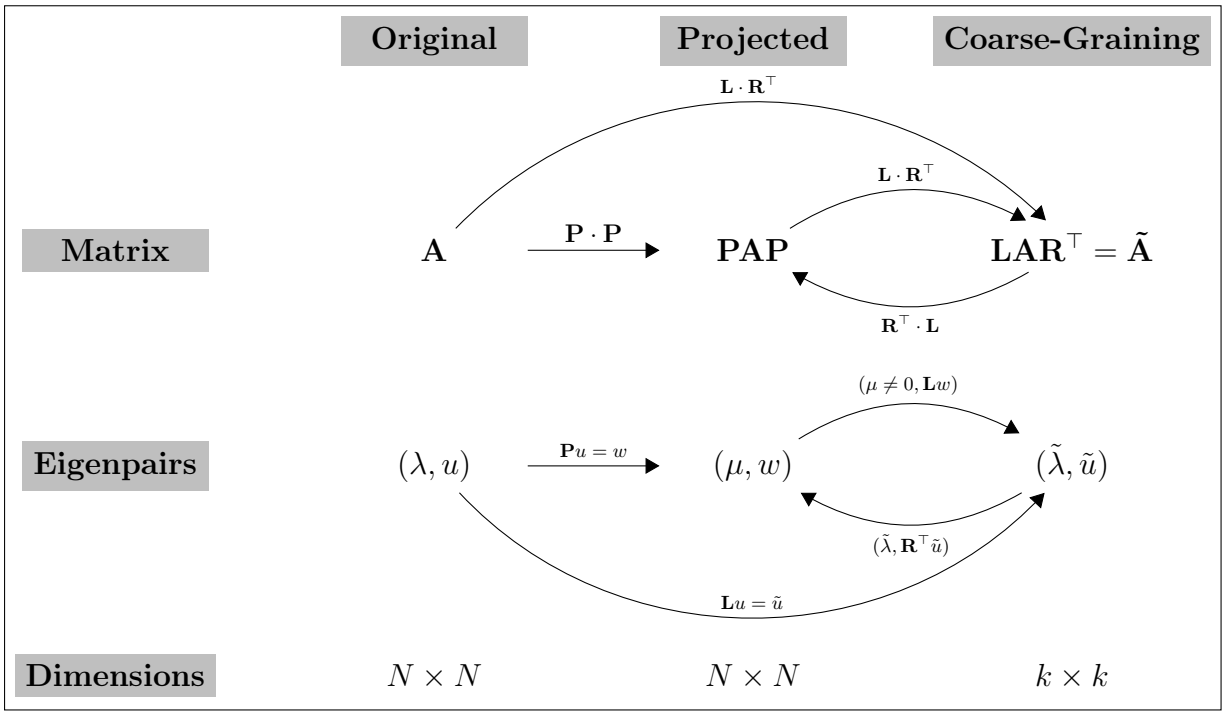


Fig. 4.2: *Summary of the relationships and transitions between the different matrices and eigenpairs in the SCG framework. Arrows indicate the direction and operation required to move from one system to another.*

The coarse-graining error for an eigenpair (λ, u) can be measured by the magnitude of the vector

$$e_P(u) \coloneqq u - \mathbf{P}u.$$

Iff $\|e_P(u_i)\| = 0$, the coarse-graining is said to be *exact* for the eigenpair (λ_i, u_i) and the following equalities hold according to Proposition 3.7 in de Lachapelle et al. (2008).

$$\begin{aligned}\tilde{\lambda}_{\gamma(i)} &= \lambda_i \\ \tilde{u}_i &= \mathbf{L}u_i \\ u_i &= \mathbf{R}^\top\tilde{u}_i\end{aligned}$$

When $\|e_P(u_i)\| > 0$, the coarse-graining is said to be *approximate* for the eigenpair (λ_i, u_i).

Finding the optimal spectral coarse-graining in respect to a single eigenpair (λ_i, u_i) can be framed as a optimization problem to find a projector $\mathbf{P}$ that solves

$$\min_{P\in\{P_0,P_1,\ldots\}} \|e_P(u_i)\|. \tag{53}$$

In absence of constraints, $\mathbf{P}$ is optimal when $\mathbf{A}$ is projected onto the eigenvector u_i:

$$\mathbf{P} = u_i u_i^\top, \tag{54}$$

This projector preserves the chosen eigenpair exactly, i.e. $\|e_P(u_i)\| = 0$. The error $\|e_P(u_j)\|$ for all $j \neq i$ is uncontrolled however.

Projector Constraints: The semantics of an adjacency matrix are not preserved with the projector defined as in (54). The resulting coarse-graining is exact but non-zero interactions between vertices that are not adjacent to each other can occur. To preserve adjacency semantics and to ensure that the relationships depicted in Figure 4.2 hold, de Lachapelle et al. (2008) impose two constraints on $\mathbf{P}$.

The *partitioning constraint* restricts each vertex of a graph to belong to exactly one group, i.e., the projector must not mix components of non-interacting groups. This constraint is formalized as

$$\forall i, j \in \{1, \ldots, N\} : \quad \gamma(i) \neq \gamma(j) \Rightarrow P_{ij} = 0 \tag{55}$$

The *homogeneous mixing constraint* ensures that vertices belonging to the same group are indistinguishable from each other in the projected system. Therefore, the same system can be represented with a smaller matrix that represents only the interaction between the groups. Formally, for any vector $x \in \mathbb{R}^N$:

$$\forall \alpha \in \Gamma, \forall i, j \in \nu(\alpha) : \quad (\mathbf{P}x)(i) = (\mathbf{P}x)(j). \tag{56}$$

The optimization problem in (53) for a single eigenpair (λ, u) is equivalently expressed by minimizing the square of $\|e_p(u)\|$. Expanded this reads as follows:

$$\|u - \mathbf{P}u\|^2 = \sum_{\alpha \in \Gamma} \sum_{i \in \nu(\alpha)} [u(i) - (\mathbf{P}u)(i)]^2. \tag{57}$$

Due to homogeneous mixing, that is, Equation (56), the subtrahend must have the same value for all components belonging to the same group. Under this constraint Equation (57) is minimal when the subtrahend $(\mathbf{P}u)(i) = \frac{1}{|\alpha_{\gamma(i)}|} \sum_{j \in \nu(\alpha)} u(j)$, i.e., the subtrahend is replaced with the average value of the eigenvector components belonging the group α;

$$\|e_p(u)\|^2 = \sum_{\alpha \in \Gamma} \sum_{i \in \nu(\alpha)} [u(i) - \frac{1}{|\alpha|} \sum_{j \in \nu(\alpha)} u(j)]^2. \tag{58}$$

Expressed in words; the minimizer projects all components onto the average (or barycenter) of their group. This minimizes the error for each group and by extension the error sum. The same term appears in various statistical contexts where it is sometimes called *within-group sum of squares* or *within-cluster sum of squares.*

Equation (58) represents the optimal projector and it is fully defined for a given partitioning Γ, i.e. the minimizing projector that satisfies the homogeneous mixing constraint is

$$P_{ij} = \frac{1}{|\alpha_{\gamma(i)}|} \delta_{\gamma(i)\gamma(j)}, \quad i, j \in \{1, \ldots, N\} \tag{59}$$

where $\delta_{\gamma(i)\gamma(j)}$ is 1 when $\gamma(i) = \gamma(j)$ and 0 otherwise. The partitioning constraint in Equation (55) is observed by this definition as well. The components of the semi projectors for real and symmetric adjacency matrices are then given by

$$R_{ij} = \frac{1}{\sqrt{|\alpha_i|}} \delta_{i\gamma(j)}, \quad i \in \{1, \ldots, k\} \text{ and } j \in \{1, \ldots, N\} \tag{60}$$

which defines matrix $\mathbf{R}$. Due to the symmetry of $\mathbf{A}$ the other semi-projector is

$$\mathbf{L} = \mathbf{R}. \tag{61}$$

In conclusion, the accuracy of the minimizing projector under the partitioning and homogeneous mixing constraints only depend on the quality of partitioning Γ. The original adjacency matrix $\mathbf{A}$, by definition, only contains components with values zero or one and has a zero diagonal. However, matrices $\mathbf{PAP}$ and $\tilde{\mathbf{A}}$ contain non-negative real values and non-zero diagonals. This corresponds to a graph with non-trivial edge weights and self-loops.

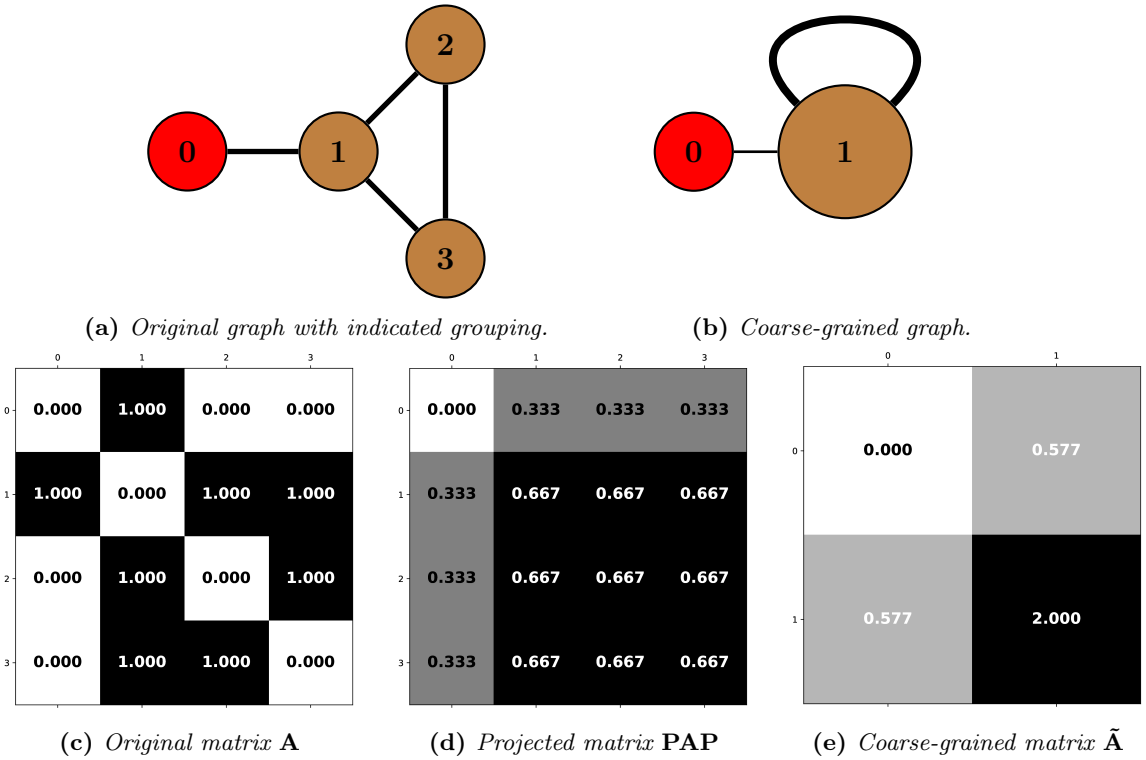
4.2.2 Spectral Coarse-Graining Example

The spectral coarse-graining for a given partitioning Γ is depicted in Figure 4.3. In $\mathbf{PAP}$ (Figure 4.3d) the homogeneous mixing constraint can be observed. All rows (and columns) belonging to the same partition have the same value given by the average value of the components of $\mathbf{A}$ that belong to the same group. This corresponds to (59). The coarse-grained graph (Figure 4.3b) represents only the group interactions. The corresponding adjacency matrix in Figure 4.3e is a weighted matrix representing the relative importance of the edges. Notice that it contains a self loop with weight 2 to represent the internal links of the original graph. A random walker entering the corresponding group in the original graph has a high probability to remain within the group by randomly following edges at each vertex. The self loop represents a combined probability to remain inside the group. The probability of going back to Vertex 0 is relatively low with weight 0.577.

4.3 Partitioning of Eigenspaces for Coarse-Graining

The minimizing projector that maximally preserves a single eigenpair (λ, u) is induced by a grouping Γ. Any eigenpair of $\mathbf{A}$ can be preserved and the choice depends on the application. The challenge is to choose a good partitioning Γ such that the term in Equation (58) is minimized. The optimum is the partitioning of u such that the sum of errors for each partition is minimal.

To help understand the error $\|e_P(u)\|^2$ of a partitioning-induced projector $\mathbf{P}$ as defined by Equation (58) an example is shown in Figure 4.4. Remember that a graph matrix of N dimensions is projected onto a k-dimensional subspace. This is best seen in Figure 4.4b where the two leading eigenvectors, that consist of $N = 198$ components each, are approximated by a rank $k = 10$ projector, that is optimized to preserve the leading eigenpair (λ_1, u_1). The leading eigenvector (blue solid line) is well approximated with only 10 components of the corresponding coarse-grained eigenvector (blue triangles). However, the partitioning is not optimized for any other eigenvector and in the example of the second leading eigenvector (magenta solid line), a large approximation error is visible (magenta diamonds). This shows, that only the preserved eigenpairs can be expected to be well approximated. A similar situation is seen with the eigenvalues in Figure 4.4a. The leading eigenvalue and its coarse-graining (blue triangles) are almost equal. The second leading



(a) *Original graph with indicated grouping.* **(b)** *Coarse-grained graph.*

(c) *Original matrix* $\mathbf{A}$ **(d)** *Projected matrix* $\mathbf{PAP}$ **(e)** *Coarse-grained matrix* $\tilde{\mathbf{A}}$

Fig. 4.3: *A coarse-graining example on a small graph. The indicated partitioning in (a) corresponds to* $\Gamma = \{\{0\}, \{1, 2, 3\}\}$*. (e) is the weighted adjacency matrix of the coarse-grained graph depicted in (b).*

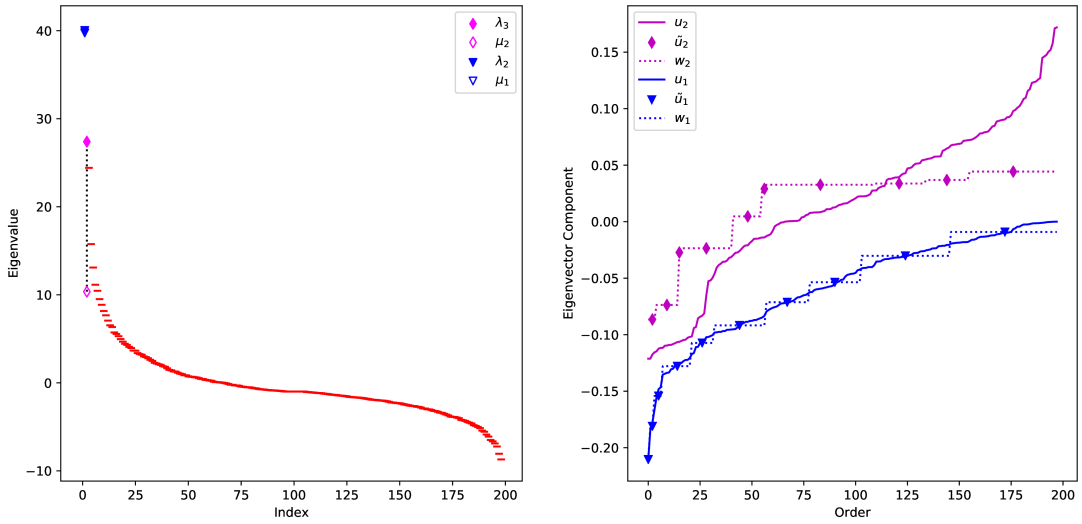
eigenvalue has a large error (magenta diamonds). This example is a good projector for the leading eigenpair but the error for other eigenpairs is not controlled

The next sections define graph partitioning via the partitioning of eigenvectors which is equivalent to vertex grouping. Then, approximation methods for solving the SCG minimization problem (53) are introduced and extended to the preservation of multiple eigenpairs.

4.3.1 Equivalence of Vertex Grouping and Eigenvector Partitioning

In Section 2.3.4 the association of vertices to eigenvector components has been defined. Given any non-zero eigenpair (λ, u) of $\mathbf{A}$. Each vertex v_i is projected onto the eigenspace of λ at coordinates given by each eigenvector component $u(i)$. Zero eigenpairs are a defective case for which above relationship does not hold. In particular, there is a zero eigenvalue for each duplicate row in $\mathbf{A}$. These rows are eliminated in any coarse-graining. It is best not to choose zero eigenpairs for preservation as their eigenvalues are not guaranteed to be preserved by optimizing $\|e_P(u)\|$.

Using the relationship described above, a partitioning of eigenvector $u \in \mathbb{R}^{\mathbb{N}}$ induces a partitioning $\Gamma = \{\alpha_1, \ldots, \alpha_k\}$, i.e., an assignment of the eigenvector components to exactly one partition $\alpha \in \Gamma$. This is equivalent to a partitioning of graph vertices. For



(a) *Eigenvalues ordered by value.* **(b)** *Eigenvector components ordered by size.*

Fig. 4.4: *Effect of a fixed-size interval partitioning on the eigenspace of the* jazz *graph. The partitioning parameters are* $e = 1$ *and* $b = 10$. *(a) Eigenvalue errors. The preserved eigenvalue* λ_1 *is well approximated by the projected eigenvalue* μ_1 *(blue triangles). The uncontrolled eigenvalue* λ_2 *has large error compared to the projected eigenvalue* μ_2 *(magenta diamonds). (b) The projector optimizes* $\|e_P(u_1)\|$. *The preserved eigenvector* u_1 *(blue line) is well approximated by the projected eigenvector* w_1 *(blue dots). The error for* u_2 *(magenta line) is uncontrolled and large.*

example, Let $u = (\kappa_1, \kappa_2, \kappa_3)^\top$ be an eigenvector and $V = \{v_1, v_2, v_3\}$ the vertex set. Then the the following partitionings are equivalent,

$$\{\{\kappa_1, \kappa_2\}, \{\kappa_3\}\} \iff \{\{v_1, v_2\}, \{v_3\}\} = \{\alpha_1, \alpha_2\} = \Gamma.$$

Partitioning (or grouping) eigenvector components and partitioning vertices is used synonymously in the remainder of this work as one can be expressed in terms of the other.

A naive approach to find the optimal partitioning is to generate and evaluate all possible partitions of u. This is infeasible as the number of distinct partitions of a set with N elements is a Bell number which grows exponentially in N. When the number of partitions k becomes a fixed parameter, the problem reduces to finding the best partitioning of u into k partitions. However, the best k, i.e., the k corresponding to the global minimizer, is not known in general. De Lachapelle et al. (2008) propose a dynamic programming approach to compute the optimal solution parameterized by k in $O(kN^2)$ time. It is not covered here due to its complexity.

4.3.2 Fixed-size Intervals Method

For large graphs, approximation methods are recommended. A simple partitioning method defined in de Lachapelle et al. (2008) is to divide u into b intervals of fixed size. Let

$\kappa_{\max} \coloneqq \max_{i\in\{1,\dots,N\}} u(i)$ and $\kappa_{\min} \coloneqq \min_{i\in\{1,\dots,N\}} u(i)$ denote the maximum and minimum component of u respectively. Then, $\delta_u \coloneqq \kappa_{\max} - \kappa_{\min}$ is the range of component values. The fixed size partitioning scheme defines each interval to be of equal size $\beta \coloneqq \delta_u / b$. The following equations define the intervals as sets $I_1, \dots, I_b$, each containing the components that fall into the respective interval:

$$
\begin{aligned}
j \in \{1, \dots, b-1\} : \qquad & I_j \coloneqq \left\{ \kappa_i : \left(\kappa_{\min} + \sum_{m=1}^{j-1} \beta \right) \leq \kappa_i < \left(\kappa_{\min} + \sum_{m=1}^{j} \beta \right) \right\} \\
j = b : \qquad & I_b \coloneqq \{ \kappa_i : \kappa_{\max} - \beta < \kappa_i \leq \kappa_{\max} \}
\end{aligned}
$$

Finally, only the $k \leq b$ non-empty sets constitute the partitioning

$$\Gamma \coloneqq \{ I_j : \forall j \in \{1, \dots, b\} \ |I_j| > 0 \}. \tag{62}$$

Figure 4.5 demonstrates this partitioning on an example vector. The parameter b determines an upper bound for the size of the coarse-graining, i.e., $k = |\Gamma| \leq b$. The lower bound is not controlled and determined entirely by the distribution of the eigenvector components. In the example this is demonstrated for I_4, which is left empty and $k = 4 < b = 5$ as a consequence. Assuming no other eigenpair is relevant, a small k is desirable and dropping empty intervals does not increase the error in (58).

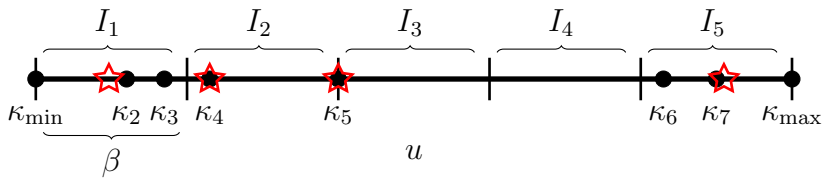


Fig. 4.5: *Example partitioning using the fixed-size intervals method. The eigenvector u is divided into $b = 5$ intervals of size β. The interval sets are $I_1 = \{\kappa_{\min}, \kappa_2, \kappa_3\}$, $I_2 = \{\kappa_4\}$, $I_3 = \{\kappa_5\}$, $I_4 = \emptyset$, $I_5 = \{\kappa_6, \kappa_7, \kappa_{\max}\}$. All non-empty intervals constitute $\Gamma = \{\{\kappa_{\min}, \kappa_2, \kappa_3\}, \{\kappa_4\}, \{\kappa_5\}, \{\kappa_6, \kappa_7, \kappa_{\max}\}\}$. The red stars markers indicate the average component value for each partition.*

Fixed-size interval partitioning makes no attempt to optimize the interval borders to the distribution of the components. The projector $\mathbf{P}$ averages all partitions and the projected eigenvector $\mathbf{P}u$ consists of only k distinct values. This is how (58) is defined. A visual example is seen in Figure 4.5 where the star markers indicate the partition average that become the coarse-grained vector.

4.3.3 K-Means Clustering Methods

An efficient and more accurate solution can be found with k-means clustering. A state-of-the-art clustering algorithm which is widely used in practice (Raghupathi, 2018). K-means iteratively improves upon an initial partitioning of N elements into k groups.

Each iteration finds a better partitioning with k groups or makes no progress and thus converges on a minimum. In general, the solution is not optimal and usually represents a local minimum.

The partitioning is performed according to *Lloyd's algorithm* (Lloyd, 1982). Its cost function is the within-cluster sum of squares, i.e. Equation (58). Section 4.2.1 demonstrates that this yields the minimizing projector for a given partitioning. Therefore, k-means can never make an initial state worse. Two methods to partition eigenvectors for SCG using k-means are presented below.

Fixed-Size Intervals + K-Means: This is the preferred method of de Lachapelle et al. (2008). K-means is used to improve an initial partitioning made with the fixed-size intervals method and defined by (62). This initialization determines that $k = |\Gamma| \leq b$; without control of the lower bound because the method parameter is b. Empty intervals can occur as in the fixed-size intervals method. The number of clusters does not change anymore after initialization. K-means creates exactly k partitions in each subsequent iteration.

Iterating with k-means only a few times optimizes an initial partitioning (62) by adjusting the size of each interval to better fit the distribution of the eigenvector components. Figure 4.6 presents an illustration of such an iteration. Notice how each interval I_j has a different size β_j now. The empty interval I_4 has disappeared because there is no corresponding mean value. Therefore, k-means ignores it. The iteration moves the interval borders such that κ_3 now lies in I_2 instead of I_1. The mean values for the affected groups are updated.

Applying k-means to the result of the fixed-size partitioning method can significantly decrease the coarse-graining error but the result is rarely optimal. The initial partitioning is ignorant of the actual distribution of the eigenvector components and tends to trap k-means in a local minimum.

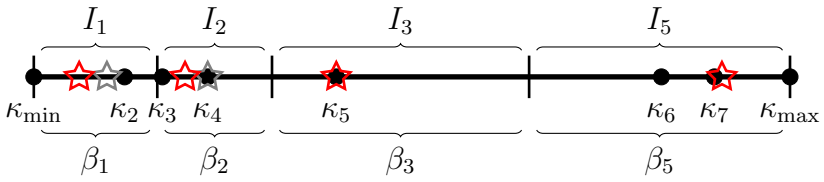


Fig. 4.6: *Example partitioning for the fixed-size intervals + k-means method. The initial partitioning is depicted in Figure 4.5. A subsequent k-means iteration adjusts each non-empty interval border to yield a lower error in* (58)*. The updated partition mean values are indicated with red star markers and the initial mean values with gray stars markers when different.*

K-Means++ Initialization: Gfeller and De Los Rios (2007) and Zeng et al. (2018b) have proposed variants of using k-means without any fixed-size interval initialization. As k-means is widely used in practice there are various in initialization methods available. In this work *k-means++* by Arthur and Vassilvitskii (2007) is used. It chooses better

initial partitions and avoids some of the problems that can adversely affect k-means. This does not fundamentally change the way the partitioning works. Therefore, most of the arguments presented for fixed-size interval + k-means are also valid for this method.

However, there is one important difference. Due to the k-means++ initialization, k is always equal to the method parameter b and empty intervals do not occur as long as $b \leq N$. Precise control of the size of the coarse-grained matrix is possible as a consequence. This difference is crucial in Chapter 6. K-means enforces a partitioning into k clusters even when it does not fit the distribution of the graph vertices. Whereas the fixed-size intervals method can be said to adjust the number of partitions to avoid creating too many groups.

Summary: Three approximation methods to find a partitioning for SCG have been presented. The *fixed-size intervals* method is the most simple of them. Its partitions can be improved by applying k-means clustering; which is called *fixed-size interval + k-means* method. Both of these methods are unable to control the size of the resulting coarse-graining precisely. In contrast, using k-means with the *k-means++ initialization* method allows precise control over the size of the coarse-graining. The first method will not be evaluated but it is used in Section 4.3.5 to derive bounds on the coarse-graining error. Those bounds are applicable to both of the other methods as well.

4.3.4 Partitioning Multiple Eigenvectors

The preservation of only a single eigenpair can lead to a large error for other eigenpairs. Most applications have heuristics to rank eigenpairs by importance but it is not always acceptable to ignore the error in all but one eigenpair. Consider spectral graph visualization which uses the eigenvectors to compute a layout for the placement of graph vertices in a multi-dimensional space. Commonly this is at least a two-dimensional problem. The vertices need to be well-separated in a rectangular image for printing or rendering. Such an application requires a projector that preserves a subspace of the eigenspace. Two methods to preserve $e \in \mathbb{N} : 1 < e \leq N$ arbitrarily chosen eigenpairs of $\mathbf{A}$ are presented in this section. Regarding zero eigenpairs, the same considerations apply as in Section 4.3 and they are best not chosen.

Segmentation Method: De Lachapelle et al. (2008) propose to compute e separate partitionings $\{\Gamma_1, \ldots, \Gamma_e\}$ where each Γ_i solves the optimization problem (53) for each eigenpair (λ_i, u_i) independently. A final partition Γ is then obtained by grouping only the vertices together that are grouped together in each Γ_i. This represents a segment of the eigenspace. This process is explained visually in Figure 4.7 in two dimensions. However, the approach extends to arbitrary dimensions and is a generalization of the fixed-size interval method.

The segmentation method is simple and flexible. Any of the methods introduced in the previous section can be used to partition the individual eigenpairs and the methods and parameters used can differ for each. Certain eigenpairs can be prioritized with a more precise approximation if they are deemed more important to an application. Even

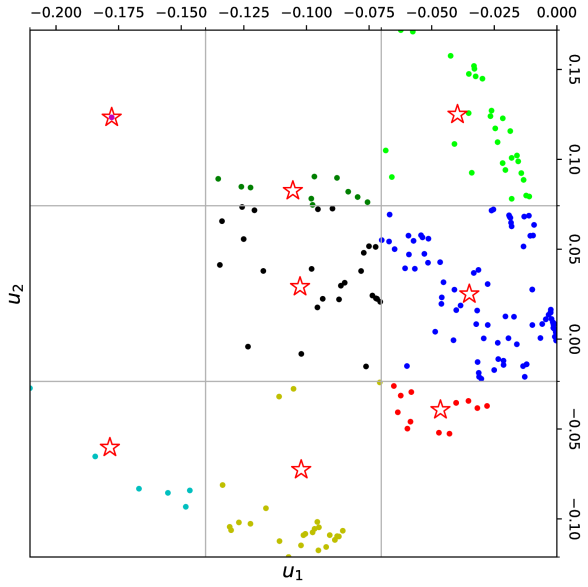


Fig. 4.7: *A partitioning of two eigenvectors ($e = 2$) of the* jazz *graph. The fixed-size interval method is used. The scatter plot displays the placement of vertices in this space as given by their eigenvector components. First, each eigenvector is partitioned into $b = 3$ intervals independently. This induces two partitionings: Γ_1 and Γ_2. The corresponding interval borders are indicated as gray lines. Second, all vertices that are grouped together in Γ_1 and in Γ_2 fall into one of the segments. Each (non-empty) segment becomes a partition in the final grouping Γ. The average (centroid) of each group is indicated with a red star marker.*

when each Γ_i is optimal, the final partition cannot be expected to be a global optimizer. Each eigenpair is considered individually when partitioning and the global error is not tracked. Generally, the vertices are not equally distributed along each eigenvector and the individual partitionings only fit the eigenvectors they were optimized for.

Because it does not optimize the global error, this method can also not control the size k of the coarse-grained matrix with the parameter b. An explosion of partitions can occur, i.e. $k \gg b$. Consider e eigenpairs, each divided into b_i intervals for $i \in \{1, \ldots, e\}$. The eigenspace is divided into $\prod_{i=1}^{e} b_i$ segments. When the vertices are distributed over most of the eigenspace, then most of these segments contain some vertices and k is large. This problem can be seen in Figure 4.8a. The lower bound depends on the methods used to partition each eigenvector. A $k < \min_i b_i$ can only occur, if at least one interval is empty along all dimensions. There are two contrary effects to consider:

- As the number of dimensions e increases, there are more intersecting dimension and it becomes more likely that a vertex falls into an interval along at least one dimension. The lower bound is driven towards b.
- As b increases, "thinner" segments are created and the chance of an interval remaining empty increases.

This makes it difficult to influence the size of the coarse-grained graph with parameter b and given the individual partitionings Γ_i for each eigenpair, k is bounded by

$$1 \leq k \leq \min\left(\prod_{i=1}^{e}|\Gamma_i|, N\right).$$

Obviously this bound is not useful as it contains all possible groupings from only one global group to each group consisting of a single vertex. An empirical evaluation in Section 5.5 determines how b and k relate on so-called "real world" data.

K-Means for Multiple Eigenvectors: K-means with k-means++ initialization can solve multi-dimensional clustering problems. This produces a single partitioning Γ that preserves e eigenpairs and minimizes the sum of individual eigenpair errors

$$\sum_{i=1}^{e}\|eP(u_i)\|^2 = \sum_{i=1}^{e}\sum_{\alpha\in\Gamma}\sum_{i\in\nu(\alpha)}[u_i(i) - \frac{1}{|\alpha|}\sum_{j\in\nu(\alpha)}u_i(j)]^2. \tag{63}$$

The optimization of the global error is sometimes preferable over the optimization of individual errors that is possible in the segmentation method.

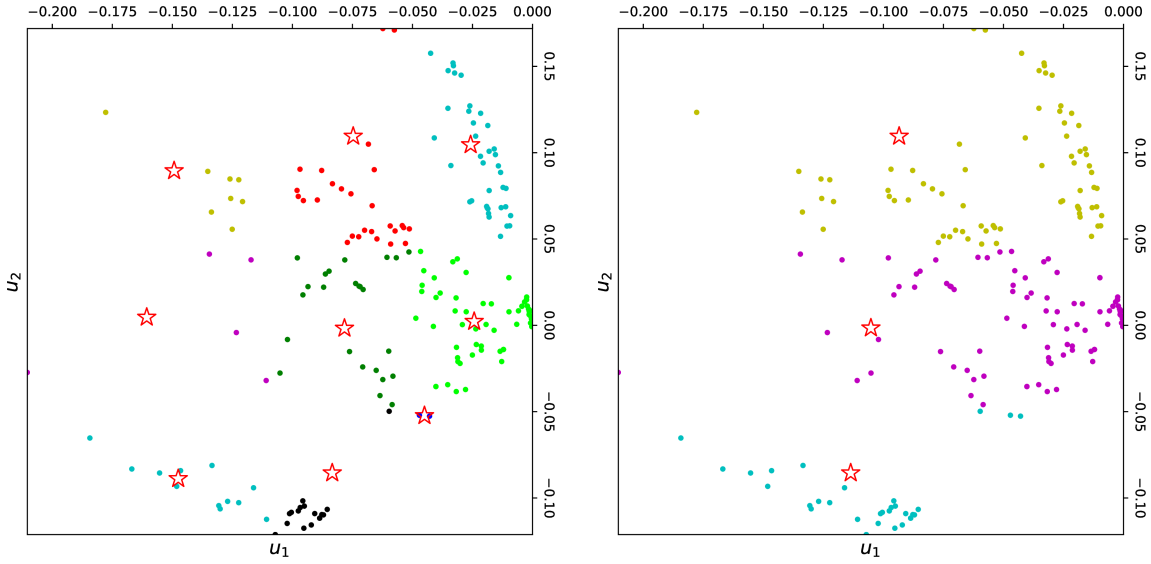
K-means does not suffer from the uncontrollable coarse-graining size problem and produces exactly the desired number of partitions $k = b$; provided there are at least b vertices. But without prior inspection of the vertex distribution in the eigenspace the optimal value for k is unknown. An uninformed choice of the parameter b is likely to be suboptimal. Compare the groups in Figures 4.8a and 4.8b. On the left, the number of partitions exploded. But it is also obvious that it fits the vertex distribution better than the figure on the right, where the number of chosen partitions is too small to adequately approximate the vertex distribution.

A partitioning with k-means has the the tendency to converge on local minima and does not allow for different parameters b for each individual eigenpair. This can be a problem when an application requires fine-grained control over individual errors and needs to approximate the leading eigenpairs with more precision.

Summary: The choice of the partitioning method is a trade-off between optimizing the global error and controlling the preservation of each eigenpair with uncontrollable size reduction. Neither method is optimal. The ability to control the number of final partitions can also be an important factor. Generally the seconds approach results in a more accurate partitioning but tends to restrict the ability to shrink the network as the number of partitions tends to grow large as a consequence of how the segmentation method combines intervals.

4.3.5 Error Bounds for Eigenvector Partitioning

Any partitioning of an eigenvector is incurring an error $\|e_P(u)\|$ defined by the minimizer in Equation (58). Fixed-size interval partitioning does not optimize upon an initial, naive



(a) *Fixed-Size Intervals + K-Means for $e = 2$ eigenpairs with $b = 3$ each. Final partition created using the segmentation method.*

(b) *K-Means for $e = 2$ eigenpairs with $k = 3$.*

Fig. 4.8: *Two different methods to partition a two-dimensional eigen-subspace of the* jazz *graph. Both methods employ k-means to optimize the interval borders and the same parameters for b and e. Group averages (centroids) are indicated with red star markers. (a) K-means is used on each dimension individually. The number of centroids explodes when the individual partitions are combined: $k \gg b$. (b) K-means is used on a two dimensional space and optimizes for the global error sum. The number of final partitions corresponds to the parameter $k = b$.*

partitioning. Therefore, it allows to derive bounds for $\|e_P(u)\|$ with relative simplicity. Here, the fact that the adjacency matrices contain only real values is used to improve the SCG error bounds from de Lachapelle et al. (2008).

The difference between the maximum and minimum component value of an eigenvector u is given by its value range δ_u. A fixed-size interval partitioning divides u into b intervals of size $\beta = \delta_u / b$ (c.f. Section 4.3.2). By limiting u to real numbers, $\delta_u \leq \sqrt{2}$. A proof for this bound is provided in Appendix E.2. Using this limit and following the steps of de Lachapelle et al. (2008, Equation 5.10), improved error bounds can be derived for $\|e_P(u)\|$ by making a worst-case assumption about the distance between any vertex to the group center.

Equation (58) states that the error for each group is the squared sum of distances between eigenvector components to the center of their group. The maximum error for any vertex is half the interval width, that is $\beta/2$, when vertices lie on the borders of their

interval. Extrapolating this worst-case partitioning to each vertex yields

$$\|e_P(u)\|^2 = \sum_{\alpha\in\Gamma}\sum_{i\in\nu(\alpha)}[u(i) - \frac{1}{|\alpha|}\sum_{j\in\nu(\alpha)}u(j)]^2 \leq \sum_{j=1}^{b}\sum_{c\in I_j}\left(\frac{\beta}{2}\right)^2 = \frac{\beta^2 N}{4}.$$

Substituting $\beta = \delta_u/b$ and then $\delta_u = \sqrt{2}$ and recalling that $b \geq k$ due to some intervals potentially being empty gives to following error bounds:

$$\|e_P(u)\| \leq \frac{\delta_u\sqrt{N}}{2b} \leq \frac{\sqrt{N}}{\sqrt{2}k}. \tag{64}$$

There is a trade-off associated to parameter b (or k) which determines the number and size of the intervals. As the interval count increases, the coarse-graining projector has more degrees of freedom to approximate the original eigenvector. However, a large b prevents SCG from shrinking the graph to small sizes. An assessment of (64) provides insight into some of the factors affecting this trade-off and coarse-graining accuracy:

- The error is more sensitive to the coarse-grained graph size k (and the parameter b) than the size of the original graph N. To stay below a fixed error $\|e_P(u)\| \leq \epsilon$:

$$k \geq \frac{\sqrt{N}}{\sqrt{2}\epsilon}. \tag{65}$$

- Equation (65) also means that large graphs can be shrunk more than small ones at the same error, i.e., to guarantee $\epsilon \approx 0.2$ with $N = 10^3$, the coarse-graining size needs to be $k \geq 112$. For $N = 10^5$, (65) yields $k \geq 1119$. The former shrinks the input by a factor nine and the latter by approximately ninety.
- The error decreases with $O(1/k)$, that is, reciprocally with k. This suggests that a very small k will cause large error but increasing k has a point of diminishing returns.
- The component range $0 \leq \delta_u \leq \sqrt{2}$ can affect accuracy significantly but there is no obvious way to control it. De Lachapelle et al. (2008) report $\delta_u \lessapprox 1$ as a common observation in experiments on random matrices. The observed values were often even much smaller than that.

It is worth reiterating that the bounds in (64) are worst cases. The analysis above is relevant because it is an upper bound for all partitioning methods covered in this thesis. Nevertheless, in general a lower error can be expected; especially with partitioning methods that optimize interval borders.

4.4 Conclusion

This chapter introduced the SCG framework which can be interpreted as a dimensionality reduction technique for graph matrices. It optimizes to maximally preserve a chosen eigensubspace and defines constraints that ensure that the result is still a valid graph matrix. Previously, it has been used primarily to reduce graph complexity in the field of network

synchronization. An implementation of SCG has been made available by de Lachapelle et al. as part of the *igraph*[4] network analysis package.

In Chapters 5 and 6, SCG is used to reduce the complexity of spectral graph algorithms and specifically, to reduce the complexity of a link prediction use case. In this context two aspects of SCG are important. First, coarse-grained graphs preserve the semantics of adjacency matrices. Unlike other dimensionality reduction techniques, the outcome is always a real symmetric matrix representing adjacency relationships of a graph. This means that applications which are defined for standard graph matrices can also work on coarse-grained graph matrices without having to be adapted in any major way. This aspect is used in Chapter 5 to define the approach at the core of the contributions of this thesis. Second, the SCG framework depends only on the quality of the eigenspace partitioning. All other aspects of the SCG framework are fully defined when a partitioning is given. Therefore, any distance-preserving approximation of an eigenspace can be used to induce a coarse-graining projector of high quality. This is used in Chapter 6 to solve the computational bottlenecks of the approach.

Furthermore, improved errors bounds for SCG have been presented in Section 4.3.5. In the remainder of this work they will serve to asses the influence of SCG on the link-prediction accuracy and to give an intuition on how much graphs can be coarse-grained before too much information is lost.

[4] https://igraph.org/

Part II

Main Contributions

Chapter 5

Coarse-Grained Link Prediction

This chapter defines and evaluates a new method that combines SPM link prediction with spectral coarse-graining. This approach enables beneficial trade-offs between link prediction accuracy and computational cost.

5.1 Introduction

Chapter 3 introduced the Spectral Perturbation Method for link prediction (SPM). It is a powerful link prediction method but very time consuming to compute on graphs with thousands or more vertices. As the size of graphs grows even further, results become infeasible to compute. This is a severely limiting factor for the applicability of SPM because link prediction problems can involve graphs that are at least one order of magnitude larger, for example, one of the datasets evaluated in this thesis contains more than 30 000 vertices. At this scale, SPM takes an excessive amount of time to compute a result and becomes impractical to use with conventional hardware.

Section 3.4.3 determined the computational complexity of the SPM algorithm to be $O(N^3)$ where N is the number of graph vertices. The main bottleneck is the computation of a full eigendecomposition to obtain all eigenvalues and corresponding eigenvectors of a graph adjacency matrix. No standard algorithm of a lower complexity class exists that is suitable to replace this step. A second bottleneck is the computation of the ranking matrix which has the same complexity (see Section 3.4.2). To design a more efficient SPM algorithm that overcomes both of these bottlenecks is difficult. In fact, it is unknown if a better algorithm exist at all.

In this chapter, a different approach is proposed to overcome both of these bottlenecks. The idea is to approximate the input graph with a smaller graph in order to obtain a similar result with reduced computational cost. This approach does not require to develop a better SPM algorithm and it addresses both bottlenecks mentioned above. The approximation of the input graph is done with the Spectral Coarse-Graining (SCG) framework defined in Chapter 4. The size reduction can be significant and under some conditions it can even reduce the complexity of SPM.

The primary contributions in this chapter are:

- The definition of Coarse-Grained SPM (CGSPM); a new method to approximate SPM at lower computational cost.
- An analysis of the computational cost and complexity of CGSPM.
- An extensive evaluation the prediction accuracy, parameter space, and computational cost of CGSPM.

The remainder of this chapter is structured as follows. The CGSPM approach is motivated in the context of spectral graph theory in Section 5.2. Section 5.3 defines CGSPM, explains design decisions and configuration options, and summarizes related work. The computational aspects of CGSPM are discussed in Section 5.4, including a presentation of the implemented algorithm and an analysis of its computational cost and complexity. To verify the proposed approach, CGSPM is compared to SPM in an empirical evaluation in Section 5.5. Then, the results and limitations are discussed and connected to previously presented theory and in Section 5.6. The chapter concludes with a summary in Section 5.8.

5.2 Motivation

A new coarse-grained link prediction method is presented in this chapter. It is based on the previously introduced Structural Link Prediction (SPM, Lü et al., 2015) and the Spectral Coarse-Graining (SCG, de Lachapelle et al., 2008) framework. This new method is called Coarse-Grained Structural Perturbation Method (CGSPM). It is characterized by the idea of predicting links on a coarse-grained graph with the goal to reduce computational cost.

A major motivation for CGSPM is that it requires no fundamental redesign of the link prediction method and the presented process is mostly agnostic to the semantics of the application and can be generalized to other use-cases. CGSPM applies SPM link prediction on the coarse-grained graph but in principle any combination of methods can be applied. The coarse-grained graph has the same structure as the input graph. Any algorithm that works on the input graph is also applicable to the coarse-grained graph. However, the effect of the coarse-graining on the performance of each application has to be evaluated individually. This chapter is limited to the evaluation of the previously introduced link-prediction scenario.

In an evaluation by Muscoloni and Cannistraci (2017) SPM is found to be the best performing global link prediction method but at same time, its high computational cost is demonstrated and it is criticized that global link prediction methods are only evaluated on relatively small graphs in most related work. While the aforementioned study is limited in scope, it recommends SPM as the best global method and concludes that it is more accurate on small graphs than on large graphs. For the latter, the authors recommend local link prediction methods as the applicability of SPM is limited by the high computational complexity.

To some extend CGSPM addresses these issues because it reduces the graph size. Coarse-grained adjacency matrices of N dimensions are projected onto a smaller, k-dimensional vector space. The desired outcomes are coarse-grained graphs of size $k < N$ that approximate the dominant structural features of the original graph. SPM is bounded by a complexity of $O(N^3)$ (see Section 3.4.3). Therefore, the coarse-grained link prediction in CGSPM, that is, the SPM part in isolation, is bounded by $O(k^3)$ which eliminates the bottlenecks if $O(k) < O(N)$. The coarse-graining with SCG requires only a partial eigendecomposition with a complexity of $O(eN^2)$ for $e \ll N$. Therefore, a reduction of the overall complexity can be expected. A detailed analysis is provided in Section 5.4.3.

Coarse-graining comes at the expense of information about the graph structure which affects the link prediction performance of SPM. However, there is evidence that it can be worthwhile trade-off because not all eigenpairs are equally important. A common observation about the eigenvalue distribution is that the eigenpairs associated to the largest absolute eigenvalues are often separated from the bulk of eigenvalues. This is seen in many studies, for example in de Aguiar and Bar-Yam (2005), Farkas et al. (2001) or Preciado and Rahimian (2017). The separated eigenvalues are commonly much larger than most of the other eigenvalues and Cvetković et al. (1980) describe these eigenvalues located on the edges of spectra as the most important for spectral applications. This is explained by the eigendecomposition defined in Equation (3) where eigenpairs with large $|\lambda|$ contribute strongly to the sum. When there are many redundancies in graph spectra, it should be possible to remove less significant dimensions and still explain most of the interactions. Redundancies are characterized by degeneracies, automorphisms, and repeating motifs. These properties can be related to the spectrum of graphs and have been observed in different contexts (e.g., de Aguiar and Bar-Yam, 2005, Farkas et al., 2001, MacArthur and Sánchez-García, 2009) and are the subject of graph theoretical works (e.g., Biggs, 1974, Part Three).

Support for the coarse-graining approach is found in the context of dimensionality reduction techniques based on the eigendecomposition or the singular value decomposition. The well known Principal Component Analysis (PCA, c.f. Abdi and Williams, 2010) is used in statistics and information retrieval disciplines to create low rank approximations of matrices for visual inspection or input to complex algorithms. An unconstrained SCG is equivalent to PCA (de Lachapelle et al., 2008) which suggests that it may be useful in the same scenarios and a matrix factorization based link prediction method is presented in Jiao et al. (2017) with a very similar motivation.

5.3 CGSPM Method Definition

A CGSPM computation is a process moving from an N-dimensional vector space to a k-dimensional coarse-grained domain with $k < N$. Link prediction is performed in the coarse domain and the result is mapped back onto the original domain. All elements of this process have been introduced in isolation before. The combined process is outlined in Figure 5.1 with graphs illustrating the effects after each transition.

In terms of the graph matrices the method is defined as follows. The input is a symmetric adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. It is considered to be the *observed* matrix described in Section 3.3. The following phases are computed:

1. *Eigenspace Partitioning:* A projector $\mathbf{P}$ and the semi-projectors $\mathbf{L}$, and $\mathbf{R}$ are approximated with one of the partitioning methods introduced in Section 4.3.
2. *Matrix Preparation:* The perturbation set $\mathbf{\Delta A} \in \mathbb{R}^{N \times N}$ is selected from $\mathbf{A}$ by removing a fraction p of its entries uniformly at random while respecting matrix symmetry. The unperturbed matrix is defined as $\mathbf{A^r} \coloneqq \mathbf{A} - \mathbf{\Delta A}$.
3. *Coarse-Graining:* The projectors shrink the unperturbed matrix to the coarse-grained unperturbed matrix $\tilde{\mathbf{A}}^{\mathbf{r}} \coloneqq \mathbf{L}\mathbf{A^r}\mathbf{R}^\top$ and perturbation set $\mathbf{\Delta}\tilde{\mathbf{A}} \coloneqq \mathbf{L}(\mathbf{\Delta A})\mathbf{R}^\top$ as defined by Equation (51). Both coarse-grained matrices are in $\mathbb{R}^{k \times k}$.

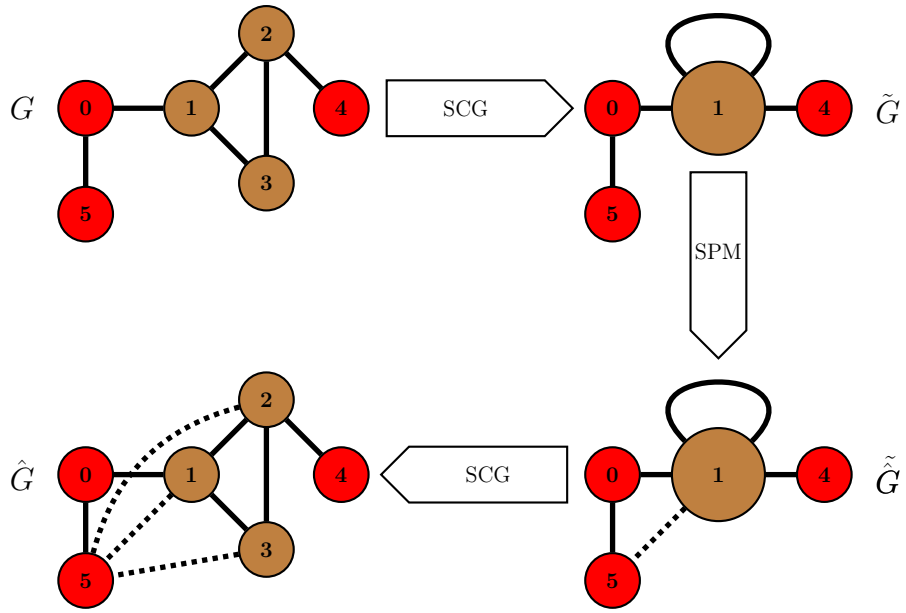


Fig. 5.1: *The CGSPM process: An input graph G is coarse-grained (upper left to upper right), then perturbed (upper right to lower right), and finally projected back to its original dimensions (lower right to lower left). Perturbations in the coarse domain (lower right) affect groups of vertices and affect all members in the original domain (lower left).*

4. *Perturbation:* For every eigenpair $(\tilde{\lambda}_i, \tilde{u}_i)$ and $i \in \{1, \ldots, k\}$ SPM computes the eigenvalue perturbation as defined by Equation (38),

$$\Delta\tilde{\lambda}_i \coloneqq \tilde{u}_i^\top \boldsymbol{\Delta}\tilde{\mathbf{A}}\tilde{u}_i. \tag{66}$$

5. *Reverse Projection:* The perturbed matrix approximation in Equation (39) is rewritten and reverse projected directly using Equation (52):

$$\hat{\mathbf{A}} \coloneqq \mathbf{R}^\top \left[\sum_{i=1}^{k} (\tilde{\lambda}_i + \Delta\tilde{\lambda}_i)\tilde{u}_i\tilde{u}_i^\top\right] \mathbf{L}. \tag{67}$$

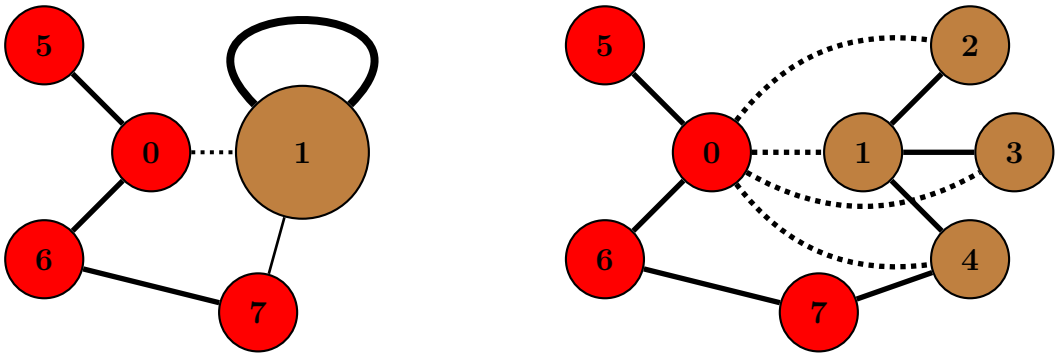
Phases 2 to 5 are iterated multiple times, to create a more robust link-existence estimator $\langle\hat{\mathbf{A}}\rangle$ by averaging all score matrices $\hat{\mathbf{A}}$ element-wise, see Equation (45). However, the SCG projectors are only approximated once as they can be re-used. This process is also formalized in Algorithm 5.2.

Method Parameters: CGSPM's performance is sensitive to the partitioning parameters e and b. They influence the ability of the projector $\mathbf{P}$ to shrink the observed matrix. A detailed discussion and explanation of their effects is given Section 4.3.

- Parameter e sets number of eigenpairs that the projector preserves in descending order by eigenvalue. Therefore, $e = 2$ preserves the two leading eigenpairs.

- Parameter b sets either the number of intervals used to partition eigenvectors or the number of final partitions in case of k-means only partitioning. In CGSPM the same value is used to partition all eigenvectors.

5.3.1 Perturbation Control under Coarse-Graining



(a) *Coarse-grained network with a small perturbation.*

(b) *An Equivalent perturbation after reverse projection of the network to the original domain.*

Fig. 5.2: *A small perturbation in the coarse-grained domain (a) can represent a large perturbation in the original graph domain (b).*

A pure black-box approach to the combination of SCG with SPM implies no knowledge of the original graph is passed to SPM and the perturbation set is chosen from the set of perturbed edges. Specifically, a fraction p of the entries in $\tilde{\mathbf{A}}$ are chosen to constitute the perturbation matrix $\mathbf{\Delta}\tilde{\mathbf{A}}$. In the original graph, a perturbation defined like this can affect many more than $p * |E|$ edges. The resulting problem is showcased in Figure 5.2. A single edge perturbation in the coarse-grained graph can represent a large perturbation in the original graph because grouped vertices cannot be distinguished. The perturbed (dotted) edge in Figure 5.2a represents four edges in the original graph (see Figure 5.2b). Without knowledge of the edges that are grouped per partition, there is no trivial way to assess the size of this perturbation. Therefore, a naive implementation can violate the fundamental assumption of SPM that the perturbation is small. This leads to the deterioration of prediction performance. The theory underlying these statements has been covered in Chapter 3. The basic assumptions of matrix perturbation theory are discussed in Section 2.5 and the impact of perturbation size on the stability of SPM is analyzed in Section 3.2.2.

An elegant solution is to choose $p * |E|$ edges from the original graph as the perturbation set (ΔE) and define the corresponding matrix $\mathbf{\Delta A}$. The unperturbed matrix $\mathbf{A^r} = \mathbf{A} - \mathbf{\Delta A}$ is prepared in the original graph domain as well. With this definition, the perturbation cannot exceed the fraction p of the edges of $\mathbf{A}$. The unperturbed matrix is coarse-grained as $\tilde{\mathbf{A}}^{\mathbf{r}} := \mathbf{L}\mathbf{A}^{\mathbf{r}}\mathbf{R}^\top$ and passed to SPM.

There are two variants to define the coarse-grained perturbation matrix $\mathbf{\Delta\tilde{A}}$.

1. Perturbed edges between vertices in the same partition can be ignored in the coarse-grained system. For any original edge $(v_i, v_j) \in \Delta E$ the corresponding coarse-grained perturbation matrix element is defined as

$$\Delta\tilde{A}_{\gamma(i)\gamma(j)} \coloneqq \begin{cases} 1 & \text{if } \Delta A_{ij} \text{ and } \gamma(i) \neq \gamma(j) \\ 0 & \text{otherwise} \end{cases} \tag{68}$$

   and its symmetric element is $\Delta\tilde{A}_{\gamma(j)\gamma(i)} \coloneqq \Delta\tilde{A}_{\gamma(i)\gamma(j)}$.
2. Perturbed edges between vertices in the same partition are represented as perturbations to the diagonal elements of $\tilde{\mathbf{A}}^{\mathbf{r}}$, i.e., the self loops of the coarse-grained vertices. Then the coarse-grained perturbation matrix is defined by the projection:

$$\mathbf{\Delta\tilde{A}} \coloneqq \mathbf{L}(\mathbf{\Delta A})\mathbf{R}^{\top}. \tag{69}$$

The second variant has been defined by the author of this work in an attempt to improve link prediction performance. It allows for a more consistent notation of the method but preliminary results shown in Figure 5.3 suggest the link prediction performance is not significantly different.

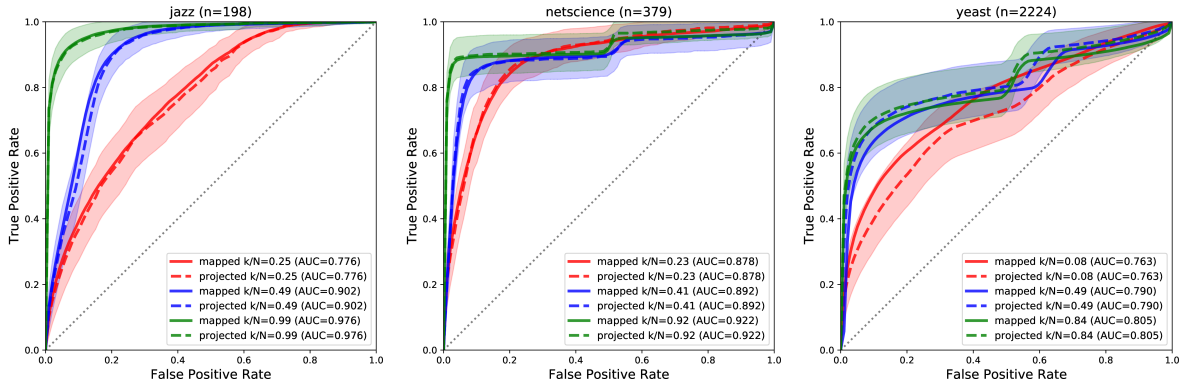


Fig. 5.3: *ROC Curves for CGSPM with projected and mapped perturbation set. The solid curve is the mapped variant and the dashed curve of the same color is the projected variant using exactly the same parameters. The shaded region is the 95% confidence interval for the projected curve and a sample size of* 30*. Plots for all evaluated graphs and more details are presented in Appendix D.6.*

5.3.2 Related Work

The robust PCA link prediction method of Pech et al. (2017, described in Section 3.3.1) shares a very similar motivation with CGSPM. In particular, the requirement for more computationally efficient global link prediction methods is addressed with of a low-rank

approximation of the graph adjacency matrix. In the case of CGSPM, SCG is used for the low-rank approximation. SCG can be interpreted as a constrained version of PCA. However, Pech et al. use the obtained low-rank matrix to compute the link-existence scores directly, while CGSPM computes a coarse-grained graph and applies SPM to it. Another loosely related line of work with similar motivation is structural link-prediction based on non-negative matrix factorization as proposed by Jiao et al. (2017).

5.4 Computational Aspects of CGSPM

This section presents the implemented CGSPM algorithms and analyzes their computational cost and time complexity. Appendix A defines the notation conventions and functions used in all following algorithms.

5.4.1 Algorithm

The CGSPM implementation of spectral coarse-graining uses the fixed-size interval + k-means partitioning method which is defined in Section 4.3 and implemented as shown by Algorithm 5.1. At Line 6 of this algorithm, the final vertex partitioning Γ is computed using the segmentation method defined in Section 4.3.4. The segmentation method is implemented in the *igraph*[5] collection of graph analysis tools. The corresponding routine is called IGRAPH_SCG_GROUPING.

Algorithm 5.1 Fixed-size + k-means SCG Projectors (SCG)

Input: $\mathbf{A}, N, e, b$

1: $\mathbf{U}_e \leftarrow \text{EIGSH}(A, e)$ ▷ *Partial eigendecomposition.*
2: $\mathcal{G} \leftarrow \{\Gamma_1, \ldots, \Gamma_e\}$
3: ***for*** $i = 1; i <= e; i \leftarrow i + 1$ ***do***
4: $\quad I_i \leftarrow \text{PARTITION}(\mathbf{U}_e(i), b)$ ▷ *Fixed-size interval partitioning, Eq.* (62)
5: $\quad \Gamma_i \leftarrow \text{K-MEANS}(\mathbf{U}_e(i), |I_i|, I_i)$
6: $\Gamma \leftarrow \text{SCGGROUPING}(\mathcal{G})$ ▷ *Segmentation method implemented in* igraph.
7: $\mathbf{R} \leftarrow \text{SCGPROJECTOR}(\Gamma)$ ▷ *Eq.* (60)
8: $\mathbf{L} \leftarrow \mathbf{R}$ ▷ *Eq.* (61)
9: ***return*** $\mathbf{L}, \mathbf{R}$

Algorithm 5.2 defines a sequential implementation of the CGSPM link prediction process which is described in Section 5.3. Its input consists of the observed graph G and a set of parameters. These parameters define the perturbation size (p), the number of perturbations (s), the number of eigenpairs to preserve (e), and the number of intervals to partition each eigenvector into (b). Refer to Sections 5.3 and 4.3 for detailed descriptions of these parameters. The result of the algorithm is a link-existence estimator $\langle \hat{\mathbf{A}} \rangle$ that assigns a score to every unobserved edge in G.

On Line 4, Algorithm 5.1 is called to obtain the SCG semi-projectors. It is executed only once because the observed matrix does not change. Therefore, the projectors can be

[5] https://igraph.org

re-used for each independent perturbation implemented as a loop on Line 7. On Lines 6 to 16, the modified SPM algorithm is implemented. Up to the SCG projections between Line 12 and Line 14, it is equivalent to Algorithm 3.2.

Algorithm 5.2 Coarse-Grained SPM Algorithm (CGSPM)

Input: $G(V, E), p, s, e, b$

1: $N \leftarrow |V|$
2: $\langle \hat{\mathbf{A}} \rangle \leftarrow \text{ZEROS}(N, N)$
3: $\mathbf{A} \leftarrow \text{A}(G)$
4: $\mathbf{L}, \mathbf{R} \leftarrow \text{GETPROJECTORS}(\mathbf{A}, N, e, b)$ ▷ *Algorithm 5.1*
5: $k \leftarrow |\mathbf{L}(1)|$
6: $e_p \leftarrow \lfloor p * |E| \rfloor$ ▷ *Fraction p of number of edges, rounded down to closest integer*
7: ***for*** $i = 1; i <= s; i \leftarrow i + 1$ ***do***
8: $\quad \Delta E \leftarrow \text{RANDOMCHOICE}(E, e_p)$ ▷ *Indices of the edges chosen as perturbation set*
9: $\quad E^r \leftarrow E - \Delta E$
10: $\quad G^r \leftarrow (V, E^r)$
11: $\quad \mathbf{A^r} \leftarrow \text{A}(G^r)$
12: $\quad \tilde{\mathbf{A}}^{\mathbf{r}} \leftarrow \mathbf{L}\mathbf{A}^{\mathbf{r}}\mathbf{R}^\top$
13: $\quad \boldsymbol{\Delta}\tilde{\mathbf{A}} \leftarrow \mathbf{L}(\mathbf{A} - \mathbf{A}^{\mathbf{r}})\mathbf{R}^\top$ ▷ *Eq.* (69)
14: $\quad \hat{\mathbf{A}} \leftarrow \mathbf{R}^\top \left[\text{PERTURB}(\tilde{\mathbf{A}}^{\mathbf{r}}, \boldsymbol{\Delta}\tilde{\mathbf{A}}, k)\right] \mathbf{L}$ ▷ *Algorithm 3.1 and Eq.* (67)
15: $\quad \langle \hat{\mathbf{A}} \rangle \leftarrow \langle \hat{\mathbf{A}} \rangle + \hat{\mathbf{A}}$
16: ***return*** $\frac{1}{s} * \langle \hat{\mathbf{A}} \rangle$

Parallelization of the Algorithm In the scope of this work, Algorithms 5.1 and 5.2 have not been optimized for any particular execution environment. However, just as the SPM algorithm (see Algorithm 3.2), they can be naively parallelized without major changes.

- The steps involved in the eigenvector partitioning, that is, in Algorithm 5.1, are partially parallelizable. However, doing so does not solve any computational bottleneck as all operations are of approximately linear complexity.
- The SCG projections are independent after the projector $\mathbf{P}$ has been obtained. All projectors can be stored in shared memory and each perturbation loop (from Line 7 to Line 14) can be parallelized equivalently as Algorithm 3.2. This is discussed in Section 3.4.1.
- The parallelization of Algorithm 3.1 (called in Algorithm 5.2, on Line 14) is discussed in Section 3.4.1 as well.

5.4.2 Operation Count

In this section, the computational cost of CGSPM is established based on an estimate of its operation count in terms of floating point operations (flop). The computational cost of SPM link prediction was estimated in Section 3.4.2. The same assumptions about the operation counts of matrix operations are used below. The following analysis has two parts. First, the computational cost is established for dense matrices and afterward, the impact of matrix sparsity is taken into consideration.

Computation Steps: The computationally relevant steps of the CGSPM implementation are described below. Omitted steps are computationally insignificant. The parameter e is the number eigenpairs that are preserved in the partitioning and b governs the intervals created for each individual partitioning. k is the uncontrollable number of final vertex groups or partitions derived by combining all partitionings of the chosen eigenspaces. The parameters are discussed in more detail in Sections 4.3 and 5.3. The only strong assumptions about parameter values made below is $e \ll N$, where N is the size of the graph. This choice is supported by the experiments in Section 5.5.

1. (Alg. 5.1, Line 1) A partial spectral decomposition is computed to obtain e eigenpairs of matrix $\mathbf{A}$. Current state-of-the-art solvers use IRLM (Calvetti et al., 1994) which is implemented, for example, in ARPACK[6]. Its operation count is given by Equation (11) but it is not predictable in detail (see Section 2.4). Therefore, only its complexity is used as a cost estimation here. As established below, it is not a bottleneck. For a dense matrix $\mathbf{A}$, the operation count of IRLM is bounded by $O(eN^2)$ flop.
2. (Alg. 5.1, Line 4) The fixed-size interval partitioning requires $e * b$ flop in total.
3. (Alg. 5.1, Line 5) K-means requires $O(bNi)$ distance calculations to partition a vector of length N into b partitions in i iterations (Hartigan and Wong, 1979). In general, i is unpredictable. In practice, it can be assumed constant and small. Therefore, a total cost of $O(ebN)$ flop is assumed for this step.
4. (Alg. 5.1, Line 6) The segmentation method is dominated by a sort of each individual partitioning. This can always be done in $O(N \log N)$ time. Even though sorting consumes execution time, no computational cost is counted because no floating point arithmetic is required.
5. (Alg. 5.1, Line 7) For each group, a matrix element for the semi-projectors has to be computed which costs k flop.
6. (Alg. 5.2, Lines 12 and 13) Each projection costs $2kN^2 + 2k^2N$ flop *per iteration.*
7. (Alg. 5.2, Line 14) The perturbation operation (Alg. 3.1) is discussed in Section 3.4.2. Assuming non-degenerate spectra, it requires $(5 + \eta)k^3 + 3k^2$ flop plus a reverse projection for $2kN^2 + 2k^2N$ flop. In total $2kN^2 + 2k^2N + (5+\eta)k^3 + 3k^2$ flop *per iteration.*
8. (Alg. 5.2, Line 15) The update to $\langle \mathbf{A} \rangle$ costs N^2 flop *per iteration.*
9. (Alg. 5.2, Line 16) The averaging for the link-existence estimator costs N^2 flop.

Steps 6 to 8 are executed s times independently. Assuming all matrices are dense and the coarse-grained spectrum is non-degenerate, CGSPM has an estimated combined cost of

$$(6sk + s + e + 1)N^2 + (6sk^2 + eb)N + s(5 + \eta)k^3 + O(k^2) \tag{70}$$

floating point operations.

Degenerate spectra only affect the SPM part of the algorithm and its cost is discussed in Section 3.4.2. Furthermore, operations in Steps 7 to 9 can be implemented more efficiently because the reverse projected matrices have a block-constant structure. However, this has not been optimized in the scope of this thesis.

[6] https://www.caam.rice.edu/software/ARPACK/

Operation Count With Sparse Matrices: The same definition of matrix sparsity as stated in Section 3.4.2 is used: sparse matrix operations have a cost proportional to the number of non-zero matrix elements (nnz). Dense matrices have $nnz = N^2$ and sparse is defined as $nnz = cN$ for a constant $0 < c \ll N$ which corresponds to $nnz = O(N)$. All graphs considered in this work are assumed to have sparse adjacency matrices and, as a consequence, $\mathbf{A}$, $\mathbf{\Delta A}$, and $\mathbf{A^r}$ are all sparse. This has the following impact on the computational cost of CGSPM.

- Step 1: The partial eigendecomposition benefits strongly from matrix sparsity (see Section 2.4). The coefficient c in Equation (11) becomes a constant and the cost is bounded by $O(e^2N)$ operations.
- Step 6: The SCG projections are sparse matrix multiplications. For the projection of $\mathbf{A^r}$, the cost is $2ck(N + k)$ and, for $\mathbf{\Delta A}$, it is $2pck(N + k)$ flop *per iteration.*
- Step 7: The matrix $\mathbf{\Delta\tilde{A}}$ is sparse but the degree of its sparsity depends on the interactions between the groups formed during the eigenspace partitioning. No strong assumption can be made about them without limiting the structure of the input graph to special cases. As a simplification, the same sparsity constant c is assumed. Therefore, the computational cost of the eigenvalue perturbation is reduced to $2pc(k + 1)$ operations per coarse-grained eigenvalue. The full cost of Step 7 becomes $2kN^2 + 2k^2N + \eta k^3 + (2pc + 4)k^2 + 2pck$ flop *per iteration.*

In conclusion, sparse, non-degenerate CGSPM has an estimated cost of

$$(2sk + s + 1)N^2 + 2sk^2N + 2s(c + pc)kN + e(e + b)N + \eta sk^3 + O(k^2) \tag{71}$$

floating point operations. The parameter p defines the fraction of links to perturb and the sparsity factor $c \approx |E|/|V|$ is given by the input graph $G(V, E)$.

Summary: In the expected regime of CGSPM, the matrix $\mathbf{A}$ is sparse and $\mathbf{\tilde{A}}$ has a nearly non-degenerate coarse-grained spectrum. The computational cost is then given in Equation (71) with k denoting the coarse-grained graph size. The relationship between N and k depends on the ability of SCG to shrink the graph. As explained in Section 4.3, the partitioning method used in CGSPM is unable to control the size k. The estimated computational cost under the assumption that k is independent of N is

$$(2ks + s + 1)N^2 + O(k^2N) \tag{72}$$

operations. The parameter s is the number of SPM perturbation. To verify this result, the relationship between k and N is evaluated empirically in Section 5.5.

5.4.3 Time Complexity Analysis

The operation count in Equation (72) does not predict algorithm runtime; only the number of operations required. While the operations count is a good indicator for the amount of computation work, it does not directly translate to a time duration. Random effects

and circumstances of computer architectures cause the runtime of individual operations to fluctuate. Additionally, operations executed in parallel reduce the overall time requirements. Therefore, time complexity in respect to graph size is studied by considering only the limiting behavior of the operation count and ignoring effects of smaller order.

The computational cost estimate established above suggests CGSPM reduces the time complexity in comparison to SPM. This conclusion assumes a significant graph size reduction due to coarse-graining, that is, a small k. However, the segmentation method which is used for eigenvector partitioning does not guarantee this (see Section 4.3.4) and the size reduction cannot be controlled reliably. This analysis proceeds in two parts. First, the time complexity is established in a favorable regime when k is small an independent of N. Thereafter, the a regime with k depending on N is considered.

For very large N, Equations (70) and (71) suggest a time complexity of $O(kN^2)$ for dense and sparse matrices. In the latter case, the eigendecomposition is not contributing to the limiting behavior anymore. The main cost for sparse CGSPM can be attributed to the reverse projection of the perturbation matrix and updates to the link-existence estimator which can both be implemented more efficiently by considering the block-constant and symmetric structure of the matrices. Theoretically, when k is sufficiently small and independent N, CGSPM can solve the eigendecomposition bottleneck of SPM.

The error bounds derived in Section 4.3.5 suggest that for coarse-graining accuracy to be maintained, k and N relate as defined by Equation (65) which means the accuracy loss is bounded by a fixed value but the method's complexity is $O(\sqrt{N}N^2)$ in this regime. Note that there is a dependency of k on N which affects the time complexity estimate. Nevertheless, this is significantly better than the time complexity of SPM. However, these error bounds concern the coarse-graining accuracy in SCG only. A more relevant performance metric for the given use-case is link prediction accuracy. Unfortunately, the relationship between SPM and SCG accuracy is complex and no error bounds for the former could be established. If an acceptable accuracy can only be achieved in a regime where the ratio k/N remains constant as N increases, that means k depends proportionally on N, then the complexity becomes $O(N^3)$.

The empirical evaluation in Section 5.5 demonstrates that CGSPM clearly reduces computational cost but the limiting behavior cannot be established with certainty. For various parameter combinations and for difficult to predict graphs, CGSPM may not always be able to achieve the theoretical complexity reduction from $O(N^3)$ to $O(kN^2)$ where k is independent of N. In the experiments, the aforementioned regime with complexity $O(\sqrt{N}N^2)$ appears to be realistic. However, it must be stressed that without knowledge of the optimal parameter e and b, the time complexity behavior is difficult to control and the search for optimal parameters is computationally demanding.

A time complexity of $O(kN^2)$ with independent k is possible when using the k-means++ initialization method for eigenvector partitioning (see Section 4.3.4). This is not evaluated with CGSPM but used with ECSPM in Chapter 6.

5.5 Experiments

The experiments and their evaluation demonstrate properties of CGSPM in realistic link prediction use-cases and provide evidence for the validity of its theoretical assessment. When the graph size is not reduced, that is, when $k = N$, CGSPM is expected to display almost equal link prediction performance and computational cost as SPM up to small differences caused by stochastic processes. This means, the metrics of SPM and CGSPM should differ more the smaller k is in relation to N. This perspective treats all metrics as functions approximating the the corresponding SPM values as k approaches N. Therefore, CGSPM link prediction performance and runtime metrics are evaluated as functions in k or in the ratio k/N throughout this evaluation.

Readers interested in comparisons of SPM to different link-prediction approaches are referred to Lü et al. (2015) and various related works (see Section 3.3.1). The procedure and data used in the following experiments are equivalent to those used by Lü et al. in the original work on SPM.

The next sections describe the evaluated data, the evaluation protocol and infrastructure. Furthermore, methods for optimal parameter choice and the measurement of link-prediction accuracy are defined. Thereafter, the results of the experiments are evaluated in three parts. The uncertain relationship between the parameters provided by an application and the coarse-grained graph size is of critical importance to the link prediction and runtime behavior. Therefore, the parameter space is evaluated first in Section 5.5.5. This establishes parameters which are subsequently used in the evaluation of link prediction accuracy in Section 5.5.6 and runtime in Section 5.5.7.

Terminology: The term "accuracy" is used informally and refers to the link prediction performance, that is, the ability to recover the hidden edges in a graph. The *accuracy metric* as defined by Equation (36) is not used in this evaluation.

5.5.1 Data Description

All datasets and data preparation steps are described in Appendix B. Graphs are referred to by their label which is defined in Table B.1. The CGSPM experiments were done on the *florida*, *jazz*, *neural*, *USAir*, *netscience*, *metabolic*, *email*, *hamster*, and *yeast* graphs.

All graphs are so-called "real world" graphs and represent interactions observed in social groups, biological systems and compounds, airplane traffic, and e-mail communication. The data was recorded by people, computer applications, or measuring equipment. Refer to Table B.1 for more details. None of the graphs is constructed from a (random) graph model. As pointed out by Lü et al. (2015), random graphs are less interesting for SPM link prediction because their structure is only predictable where it is not randomly generated.

5.5.2 Evaluation Protocol and Infrastructure

Throughout this evaluation, the link prediction problem is treated as a binary classification task. For a given observed graph, each unobserved edge is classified as either a *missing edge*

(positive class) or a *non-existing edge* (negative class). The SPM and CGSPM classifier models are used in the experiments. They are defined as follows.

- The *SPM model* is implemented according to Algorithms 3.1 and 3.2. It requires the parameters p and s. The former defines the relative size of the perturbation and the latter defines the number of independent perturbations.
- The *CGSPM model* is implemented according to Algorithm 5.2. The coarse-graining uses the *fixed-size intervals + k-means* segmentation method (see Section 4.3). The perturbation set is mapped to the coarse-grained domain as defined by Equation (68) (see Section 5.3.1). The CGSPM model requires the SPM parameters p and s and two additional SCG parameters. Parameter e defines the number of eigenpairs to preserve in the coarse-graining and parameter b defines the "resolution" of the approximation, that is, the number of intervals used to partition each individual eigenvector.

Each dataset from Table B.1 induces a graph $G^*(V, E)$ and the set E^C which is the complement of E and contains all non-existing edges between vertices in G^*. Each model, together with a set of model parameters, defines a classifier and each classifier is tested several times on each graph. A *test* of a graph $G^*(V, E)$ with a given classifier consists of two steps:

1. $\lfloor 0.1 * |E| \rfloor$, that is, approximately 10%, of the edges in E are selected independently and uniformly at random without replacement to consists a test set E^{test}. The remaining edges constitute the training set $E^{\text{train}} = E - E^{\text{test}}$ ($\approx 90\%$ of the edges).
2. The classifier is trained on the observed graph $G(V, E^{\text{train}})$ and produces a score matrix $\langle \hat{\mathbf{A}} \rangle$ as output.

Every classifier is subjected to 100 independent tests on every graph; unless specified differently.

Parameters Values: The SPM parameters are always set to the same values as in Lü et al. (2015). Those are: $p = 0.1$ and $s = 10$. The SCG parameters influence the size of the coarse-graining. As explained in Section 4.3, their exact effect is data-dependent and can not be predicted exactly.

Therefore, the CGSPM parameter space is grid-searched. The permissible values for e and b are all integers $1, \ldots, N$. As a result, each graph has a parameter space of size $N \times N$. A full exploration is infeasible because it would require more than 10^9 samples in total; each sample has a complexity of $O(kN^2)$ or larger (see Section 5.4.3). Therefore, the parameter space samples are limited to all combinations of $e \in \{1, \ldots, 10\}$ and $b \in \{10, 20, \ldots, \lfloor |V|/10 \rfloor * 10\}$, that is, from 10 to N in steps of 10. This produces a fairly dense sampling for the parameter space for b. In total, 688.000 samples or approximately N samples per graph. Due to the large amount of samples, they cannot all be reported in this evaluation. In some charts, their values are reported as unlabeled gray lines.

Infrastructure: All experiments were conducted on a cluster of 39 dedicated computation nodes. Their hardware specifications are summarized in Table 5.1. Due to the large amount of computation, resource-homogeneity was only a secondary concern. Nevertheless, results

should be comparable as all computation nodes have similar architecture and capabilities. Furthermore, each test was assigned independently to one of the computation nodes based on resource availability at the time of execution. Therefore, all reported summary statistics consist of results computed on a mix of the available computation nodes.

Table 5.1: *Computation hardware used in the experiments.*

Number	Cores	Memory	Model
20	2x10	128 GB	Intel Xeon E5-2680 v2 2.80GHz
18	2x12	128 GB	Intel Xeon E5-2650 v4 2.20GHz
1	2x12	512 GB	Intel Xeon E5-2650 v4 2.20GHz

5.5.3 Parameter Optimization

Given an accuracy constraint, the optimal parameters e and b are defined where k is minimal under this constraint. For any graph, this is a constrained optimization problem with constraints s_{lb} and s_{ub}:

$$\begin{aligned} \arg \min_{e,b \in \{1,\dots,N\}} \quad & |\Gamma(e,b)| \\ \text{subject to} \quad & s_{lb} \leq S(e,b) < s_{ub}, \end{aligned}$$

where $\Gamma(e,b)$ assigns a vertex partitioning of that graph to a set of given SCG parameters and $S(e,b)$ is a scoring function that measures link prediction accuracy, for example, the AUC or precision metric. This optimization strategy yields a parameter combination with minimal coarse-grained graph size $k = |\Gamma(e,b)|$ while respecting the bounds on the score. This strategy is referred to as the *accuracy-constrained* parameter optimization strategy and its primary purpose is to limit accuracy loss.

Given a computational constraint on the coarse-grained graph size, the optimal parameters are defined where the link prediction score difference to SPM is minimal under this constraint. For any graph, this is a constrained optimization problem expressed as:

$$\begin{aligned} \arg \min_{e,b \in \{1,\dots,N\}} \quad & S(e,b) \\ \text{subject to} \quad & k_{lb} \leq |\Gamma(e,b)| < k_{ub}, \end{aligned}$$

with constraints given as the coarse-grained graph size k. It is straight-forward to translate the constraints to relative sizes or computational cost estimates via Equation (72). This strategy is referred to as *cost-constrained* because it limits k which is used as a proxy for computational cost.

The optimization strategies demonstrate two techniques to find optimal parameters when the parameter space is known.

5.5.4 Link Prediction Accuracy Measurement

Accuracy statistics are calculated by ranking all unobserved edges according to their score in $\langle \hat{\mathbf{A}} \rangle$. The details of this ranking are described in Section 3.2. Because all matrices in this thesis are symmetric, only the upper triangle of each score matrix is considered. The link prediction performance is evaluated using receiver operator characteristic (ROC) curves that are defined and explained in Section 2.7. Each link existence estimator $\langle \hat{\mathbf{A}} \rangle$ produces a ROC curve by varying the score threshold above which edges are classified as missing edges. Any missing edge contained in E^{test} is a true positive; otherwise it is a false positive classification. All remaining unobserved edges are classified as non-existing. Non-existing edges contained in the test set are false negatives; otherwise they are true negatives. The reported ROC curves are averaged over all tests on the same graph, parameters, and classifier. ROC curve averaging is done using the vertical averaging method (see Algorithm 2.1). The area under the ROC curve (AUC) is calculated with Equation (37). The true positive and true negative sets are defined for each $\langle \hat{\mathbf{A}} \rangle$ as

$$S_{tp} = \{\langle \hat{A} \rangle_{i,j} : \forall i, j \ (v_i, v_j) \in E^{\text{test}}\}$$
$$\text{and } S_{tn} = \{\langle \hat{A} \rangle_{i,j} : \forall i, j \ (v_i, v_j) \in E^{C}\}.$$

Then, all pairs between the elements of the two sets S_{tp} and S_{tn} are compared to determine the counts used in Equation (37).

When precision is reported, it is computed with Equation (35). In this case, the threshold is chosen to separate the top-$|E^{\text{test}}|$ (approximately 10%) edges ordered by their score in $\langle \hat{\mathbf{A}} \rangle$. All unobserved edges with a link-existence score above the threshold are classified as missing edges. This is the same precision metric as the one used in all SPM related evaluations (see Section 3.3.1).

5.5.5 SCG Parameter Results

The SCG parameters are e, the number of eigenpairs to preserve, and b, the number of intervals to partition each eigenvector into. Both parameters are discussed in Section 4.3. The following evaluation demonstrates their effect on the relative coarse-grained graph size, that is, the ratio k/N, and prediction accuracy.

The coarse-grained graph size is measured by recording the dimensions of the SCG semi-projector $\mathbf{R} \in \mathbb{R}^{k \times N}$ in each test. Instead of reporting the average values for k, the mode of all values obtained from the tests with the same classifier and graph is used in order to always have an integer value. The SPM model does not reduce the graph size and it can be interpreted as $k = N$.

The parameter choice determines k/N in a non-trivial and graph dependent manner. This is shown in Figure 5.4 with four representative examples. The white and bright-yellow regions of large k/N appear to dominate the parameter space. All plots are truncated in the y-axis because no samples above $e = 10$ were taken even though the whole parameter space extends up to $e = N$. However, the results suggest that the truncated part is predominantly white or bright-yellow which means no significant size reduction can be achieved.

As e and b grow large, the regions with significant size reduction become small or do not exist. There is a dominant pattern on all graphs: large size reduction occurs only close to the origin and along the axes of the parameter space. All levels of size reductions can be achieved with $e \leq 10$ by varying only b. Even with $e \leq 2$ most k can be achieved. Alternatively, b can be fixed and e varied. Regions where the transition from red to white is small, in either the e or b direction, are described as having a large or steep gradient. In those regions, small parameter changes cause large differences in k. The transition is steep on *jazz* in the e direction at $b > 60$ and for *yeast* in the b direction at $e > 3$. Steep gradients are observed on most other graphs in the evaluation. The *netscience* and *USAir* graphs show the smallest gradients overall.

Figure 5.5 reports the difference between the average AUC of SPM (μ_{spm}) and the average AUC of CGSPM (μ_{cgspm}) in the parameter space. It shows a similar pattern as in Figure 5.4 above. CGSPM accuracy performs badly (red colors) in regions where k is small and good (bright colors or gray) in regions where k is large. However, there always exist areas where the AUC performance is good and k is relatively small. In this aspect, the *USAir* graph is remarkable as CGSPM predicts its edges accurately with most evaluated parameter combinations. The *yeast* plot shows a clear loss of accuracy at the origin but otherwise the regions of high accuracy intersect with regions of considerable coarse-graining. In contrast, the *netscience* and especially the *jazz* plots show larger regions of decreased accuracy. On the *jazz* graph, most of the parameter space shows no statistically significant difference to SPM (gray area).

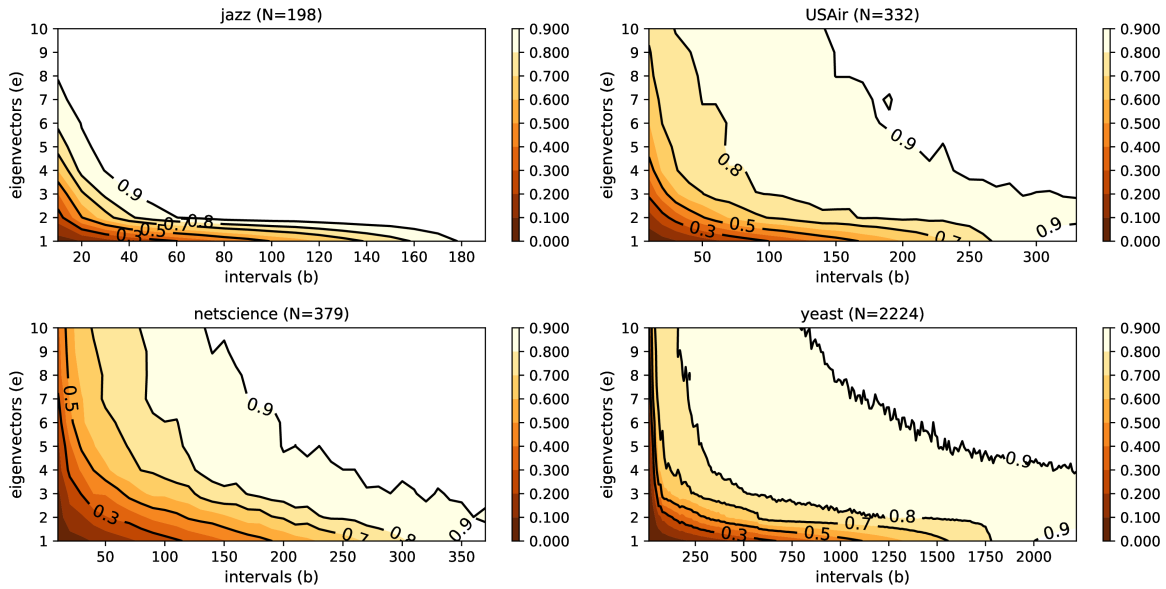


Fig. 5.4: *Relative coarse-grained graph size k/N for the sampled section of the parameter space. Dark-red colors indicate big size reduction ($k < 0.3N$) and bright regions insignificant size reduction ($k \geq 0.9N$). Plots for all evaluated graphs are presented in Appendix D.1.*

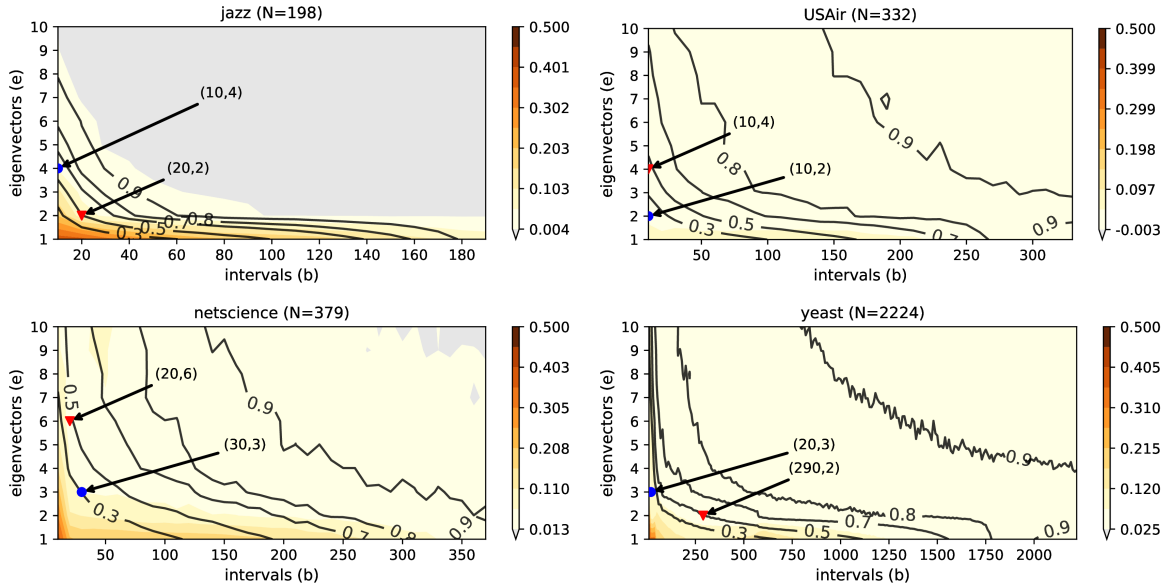


Fig. 5.5: *AUC loss compared to SPM ($\mu_{spm} - \mu_{cgspm}$) for the sampled section of the parameter space. Red colors indicate a large difference and bright colors a small difference. That means, a bright colors indicate better CGSPM link prediction performance. In gray areas, the value is not different from SPM at 95% confidence level. Black contours show k/N increasing towards the upper right (northeast). Optimal points for accuracy-constrained (blue) and cost-constrained (red) optimization strategies are annotated. Appendix D.2 shows the plots for all evaluated graphs.*

Optimal Parameters: Table 5.2 lists the optimal accuracy-constrained points in the sampled parameter space. The scoring function is defined as $S(e, b) = \mu_{\text{spm}} - \mu_{\text{cgspm}}$ and the lower bound is $s_{lb} = 0$. The upper bound s_{ub} has been chosen, arbitrarily, to be located at the tenth percentile of the interval between the upper bound of the confidence interval for $S(e, b) = 0$ and the maximum possible AUC difference. This corresponds to the most bright-yellow area in Figure 5.5. The parameters shown in Table 5.2 are annotated with blue dots in those plots. It is evident that accuracy-constrained points are located relatively close to the origin of the parameter space where k tends to be relatively small. Except for *florida*, where k is only $0.92N$ at the optimal accuracy-constrained point, the size reductions are considerable on all graphs. In particular, the *USAir* ($k = 0.16N$) and *yeast* ($k = 0.08N$) graphs stand out. This distribution of the optimal parameters around the origin is observed at different accuracy levels as well (see Appendix C.1, Table C.1).

Optimal cost-constrained parameters are listed in Table 5.3. The optimization problem is solved with $k_{lb} = 0$ and $k_{ub} = 0.5N$. This results in parameters that yield a k/N ratio that is not exceeding the 0.5 contour line in the plots of Figure 5.5. The cost-constrained points are indicated with red triangles in those plots. The optimal cost-constrained points remain close to the origin as well. None of the points is located at any extreme. The optimum in the *neural* graph at $k/N = 0.38$ is not very close to the the $k = 0.5$ contour because its coarse-graining size gradient is very steep at this point and none of the sampled

Table 5.2: *Optimal parameters constrained to maximally 10% AUC difference.*

Graph	b	e	$\frac{k}{N}$	CI*	s_{ub}	$\Delta\mu = \mu_{\text{spm}} - \mu_{\text{cgspm}}$
florida	10	4	0.92	[0, 0.0070]	0.0563	0.014
jazz	10	4	0.60	[0, 0.0041]	0.0537	0.049
neural	10	4	0.59	[0, 0.0078]	0.0571	0.030
USAir	10	2	0.16	[0, 0.0000]	0.0500	0.014
netscience	30	3	0.31	[0, 0.0141]	0.0627	0.059
metabolic	20	3	0.46	[0, 0.0082]	0.0573	0.047
email	130	2	0.53	[0, 0.0066]	0.0559	0.047
hamster	70	2	0.32	[0, 0.0036]	0.0533	0.052
yeast	20	3	0.08	[0, 0.0243]	0.0719	0.068

* The confidence interval is determined with $\alpha = 0.05$ by LS means approximation of the lower-tailed Dunnett-contrast statistic with $H_1 : \mu_{\text{spm}} > \mu_{\text{cgspm}}$ (Dunnett, 1955, Lenth, 2016).

Table 5.3: *Optimal CGSPM parameters constrained by $k/N \leq 0.5$.*

Graph	b	e	$\frac{k}{N}$	$\Delta\mu = \mu_{\text{spm}} - \mu_{\text{cgspm}}$
florida	10	2	0.41	0.210
jazz	20	2	0.49	0.073
neural	10	3	0.38	0.078
USAir	10	4	0.44	0.000
netscience	20	6	0.47	0.049
metabolic	20	3	0.46	0.047
email	110	2	0.49	0.064
hamster	40	3	0.45	0.026
yeast	290	2	0.49	0.041

parameter combinations is located close to the contour line. The *florida* graph's AUC is low at the optimal point because the 0.5 contour line never passes through a high-AUC region. Similar results can be observed at different coarse-grained graph sizes (see Appendix C.1, Table C.2).

In summary, the parameter spaces of different graphs show very different performance metrics. Nevertheless, optimal parameters distribute close to the origin.

5.5.6 CGSPM Link Prediction Accuracy Results

The theory developed in this chapter hypothesizes that the link prediction accuracy of the coarse-grained approach approximates the original SPM accuracy as k approaches N. The results presented below verify this claim by evaluating accuracy as a function in the ratio k/N. A second hypothesis is that beneficial trade-offs between link prediction accuracy and computational cost can be found. A beneficial trade-off is marked by a slow accuracy decrease as the ratio k/N becomes smaller. In other words, when accuracy decreases slower than k, the trade-off is beneficial.

The accuracy evaluation is split into two part. First, the AUC and precision metrics are reported. Then, the CGSPM classifiers are evaluated with ROC curves that allow better insights into the factors affecting link prediction performance.

Table 5.4: *Best CGSPM AUC and precision at evenly spaced k/N. Bold cells indicate the first value where SPM performance is matched (rows at the top are better). Graphs are ordered by size from smallest to largest.*

ROC-AUC Results

$\frac{k}{N}$	florida	jazz	neural	USAir	netscience	metabolic	email	hamster	yeast
0.1	0.551±0.03	0.595±0.04	0.637±0.03	0.858±0.02	0.751±0.03	0.794±0.02	0.725±0.01	0.846±0.01	0.763±0.01
0.2	0.529±0.02	0.656±0.02	0.712±0.02	0.910±0.01	0.838±0.03	0.785±0.02	0.759±0.02	0.868±0.01	0.761±0.01
0.3	0.559±0.03	0.776±0.02	0.734±0.02	0.917±0.01	0.878±0.02	0.821±0.02	0.780±0.01	0.892±0.01	0.761±0.01
0.4	0.658±0.03	0.864±0.01	0.815±0.02	0.922±0.01	0.894±0.02	0.848±0.02	0.780±0.01	0.912±0.01	0.769±0.01
0.5	0.735±0.02	0.902±0.01	0.815±0.02	**0.927±0.01**	0.899±0.02	0.879±0.01	0.816±0.01	0.925±0.01	0.788±0.01
0.6	0.735±0.02	0.902±0.01	0.863±0.01	0.930±0.01	0.898±0.03	0.880±0.02	0.838±0.01	0.936±0.01	0.801±0.01
0.7	0.735±0.02	0.934±0.01	0.871±0.01	0.932±0.01	0.898±0.03	0.908±0.01	0.859±0.01	0.946±0.01	0.807±0.01
0.8	0.858±0.02	0.954±0.01	0.884±0.01	0.927±0.01	0.901±0.03	0.914±0.01	0.872±0.01	0.950±0.00	0.803±0.01
0.9	0.874±0.02	0.967±0.01	**0.893±0.01**	0.928±0.02	0.917±0.02	0.916±0.01	**0.881±0.01**	**0.951±0.00**	0.805±0.01
MAX	**0.947±0.01**	**0.976±0.00**	0.892±0.01	0.928±0.01	0.936±0.02	**0.927±0.01**	0.881±0.01	0.952±0.00	0.804±0.01
SPM	0.945±0.01	0.976±0.00	0.892±0.01	0.924±0.01	**0.948±0.02**	0.926±0.01	0.880±0.01	0.951±0.01	**0.831±0.01**

Precision Results

$\frac{k}{N}$	florida	jazz	neural	USAir	netscience	metabolic	email	hamster	yeast
0.1	0.058±0.02	0.051±0.02	0.062±0.02	0.168±0.03	0.054±0.03	0.084±0.02	0.022±0.01	0.055±0.01	0.031±0.01
0.2	0.077±0.02	0.101±0.02	0.048±0.01	0.279±0.03	0.052±0.02	0.071±0.02	0.032±0.01	0.040±0.01	0.029±0.01
0.3	0.098±0.02	0.132±0.02	0.047±0.01	0.196±0.04	0.088±0.03	0.060±0.02	0.031±0.01	0.032±0.01	0.029±0.01
0.4	0.108±0.02	0.145±0.02	0.056±0.02	0.171±0.03	0.085±0.03	0.052±0.02	0.031±0.01	0.028±0.01	0.006±0.00
0.5	0.121±0.03	0.159±0.03	0.056±0.02	0.106±0.03	0.091±0.03	0.051±0.02	0.011±0.01	0.042±0.01	0.004±0.00
0.6	0.121±0.03	0.159±0.03	0.054±0.02	0.063±0.03	0.090±0.03	0.043±0.02	0.009±0.01	0.020±0.01	0.003±0.00
0.7	0.121±0.03	0.175±0.03	0.039±0.01	0.082±0.04	0.040±0.02	0.026±0.01	0.011±0.01	0.042±0.01	0.006±0.00
0.8	0.130±0.03	0.236±0.03	0.044±0.01	0.338±0.05	0.056±0.06	0.015±0.01	0.008±0.00	0.121±0.10	0.127±0.02
0.9	0.105±0.03	0.392±0.07	0.042±0.03	0.430±0.03	0.337±0.06	0.098±0.08	0.023±0.03	0.508±0.02	0.147±0.01
MAX	**0.548±0.02**	**0.656±0.02**	**0.167±0.02**	**0.442±0.03**	0.363±0.05	**0.348±0.03**	0.138±0.02	0.515±0.02	0.149±0.01
SPM	0.547±0.02	0.652±0.02	0.166±0.02	0.441±0.03	**0.409±0.05**	0.344±0.03	**0.148±0.01**	**0.521±0.01**	**0.171±0.01**

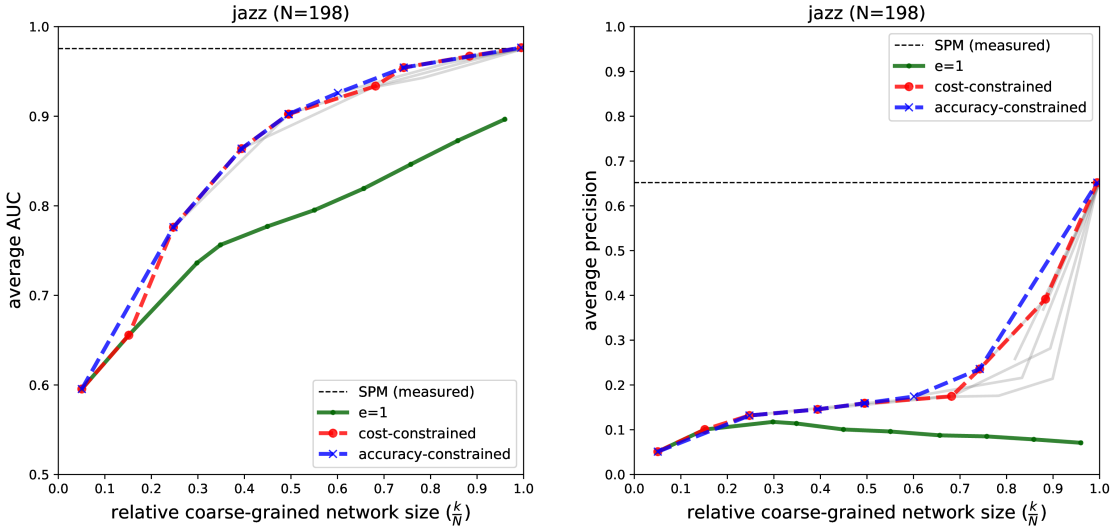
All values are reported in the format <mean> ± <standard deviation> for a sample of size 100.

AUC and Precision: Average AUC and precision values for evenly spaced values of the ratio k/N are reported in Table 5.4. For every graph, table rows indicate the best sampled average accuracy at the corresponding relative coarse-grained graph size. The values are chosen along a contour line in parameter space. Each table column contains the AUC or precision values at increasing coarse-grained graph sizes. The row labeled with "MAX" contains the maximum accuracy that was recorded. Theoretically, this value should be located where $k = N$. However, a value for $k = N$ does not always exist because only a subset of the parameter space has been sampled.

Table 5.4 shows that CGSPM reaches the AUC and precision values of SPM on most graphs. The plots presented in Figure 5.6 demonstrate the behavior of AUC and precision as functions in k/N on the *jazz* graph. The SPM performance level is indicated by a horizontal dashed line. Gray curves trace the distribution of accuracy values sampled in the evaluation. The curve for $e = 1$ is highlighted in green. The cost-constrained (red)

and accuracy-constrained (blue) parameter optimization strategies are indicated as well. The accuracy-constrained curve corresponds to the *jazz* column in Table 5.4.

The cost-constrained function is limited on the x-axis. At each sample it chooses the best accuracy below a given relative coarse-grained graph size (limiting computational cost). The accuracy-constrained function is limited on the y-axis, that means, it chooses the lowest k above a given AUC level (guaranteeing an accuracy level). In all cases, the optimization is guided by the AUC metric and does not consider the precision scores. Supplementary plots for the other graphs can be found in Appendix D, Figures D.3 and D.4. Supplementary data tables for the accuracy and cost-constrained functions are presented in Appendix C, Tables C.3 and C.4.

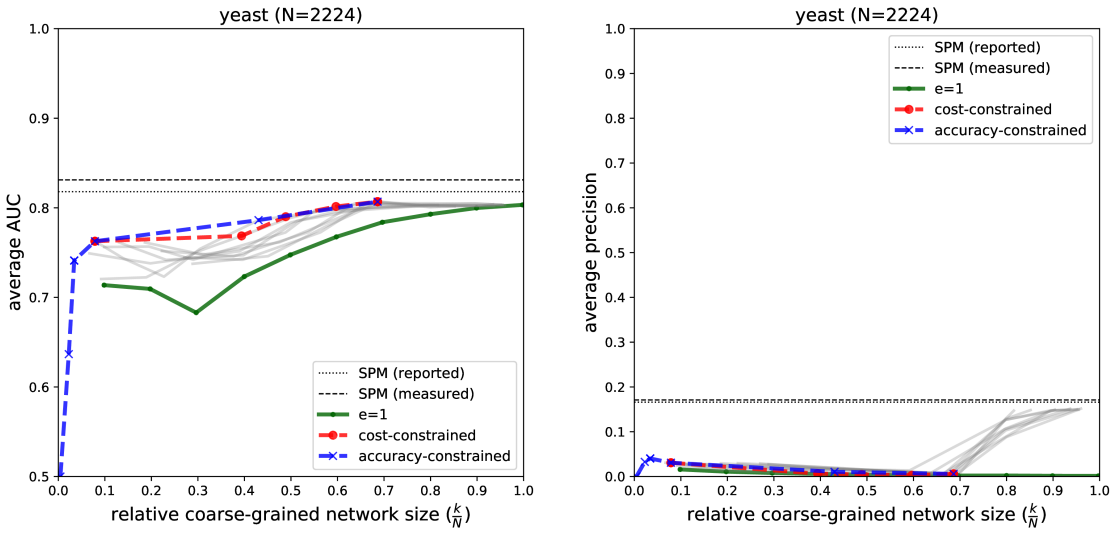


(a) *ROC Area Under Curve* **(b)** *Precision*

Fig. 5.6: *Link prediction accuracy on the* jazz *graph as a function in* k/N.

All graphs show common behaviors of the accuracy metrics that are represented in Figures 5.6a and 5.6b. Their AUC and precision functions approach the SPM level as k/N becomes large. Furthermore, there is a clear gap between the AUC and precision curves for $e = 1$ and corresponding curves for $e > 1$ (gray lines). The latter tend to cluster in the same parts the accuracy space. Finally, both optimization strategies perform very similarly on all graphs. They choose high accuracy points in AUC space which confirms that they are working as designed. Concerning the accuracy trade-offs, it can be seen that AUC curves tend to be concave with small slope in high AUC regions, that means, they allow to trade-off accuracy for lower k at a favorable rate. On the other hand, precision curves tend to be convex in high AUC regions and all precision scores drop off dramatically to very low values with decreasing coarse-grained graph size.

The *netscience*, *email*, *hamster*, and *yeast* graphs fail to reach the SPM precision level for any sampled parameter combination. Furthermore, the *netscience* and *yeast* graphs also fail to reach the SPM AUC level. The accuracy for the latter is plotted in Figure 5.7.



(a) *ROC Area Under Curve* **(b)** *Precision*

Fig. 5.7: *Link prediction accuracy on the* yeast *graph as a function of the relative coarse-grained network size k/N. Additionally to the measured accuracy level of SPM, the accuracy reported in Lü et al. (2015) is shown with a horizontal dotted line. The discrepancy is discussed in Section 5.7.3.*

The gray lines in both plots show that these graphs never reach $k = N$ in any of the sampled parameter combinations with $e > 1$.

Another interesting behavior is seen on the AUC function for the *yeast* graph. The AUC quickly rises to high levels at very low coarse-grained graph sizes, then reaches its maximum at $k \approx 0.7N$, and decreases slightly thereafter (see Figure 5.7a). Both optimization strategies stop at this maximum. As a consequence, the precision incline seen in Figure 5.7b at $k > 0.7N$ is completely missed by both strategies. The same happens on the *USAir* graph. However, the AUC on *USAir* surpasses the SPM level.

More support for favorable trade-offs between AUC and k can be found by considering the values in Table 5.5. It contains the relative coarse-grained graph sizes (k/N) at increasing AUC differences ($\Delta\mu$) relative to the AUC of SPM. Each cell reports the minimal (best) k/N at which the corresponding accuracy is achieved, that means, the columns contain the x-axis values of the accuracy-constrained AUC function. All graphs, except *florida*, can be shrunk by more than 10% without exceeding 10% relative AUC loss. At 20% AUC loss, these graphs can be shrunk by at least 50%. This corresponds to a 5:2 size to accuracy trade-off; on most graphs even substantially better. However, there are large differences between different graphs. Most importantly, interesting trade-offs exist in the high-AUC regions on most evaluated graphs. The *metabolic* and *email* graphs can be shrunk by approximately 35% without losing more than 5% AUC. The *hamster* and *yeast* graphs can be shrunk by at least 55% while staying at the same relative accuracy level. And the *USAir* graph achieves this accuracy level at almost 85% size reduction.

Table 5.5: *Relative coarse-grained graph sizes for the accuracy-constrained function.*

	$\frac{k}{N}$								
$\Delta\mu$%	florida $N = 128$	jazz $N = 198$	neural $N = 297$	USAir $N = 332$	netscience $N = 379$	metabolic $N = 453$	email $N = 1133$	hamster $N = 1788$	yeast $N = 2224$
MAX	1.00	0.99	**0.91**	**0.63**	0.98	1.00	0.96	**0.93**	**0.69**
≤ 5	0.92	0.74	0.59	**0.16**	0.85	**0.62**	0.65	**0.45**	**0.43**
≤ 10	0.92	0.60	0.59	**0.16**	**0.31**	0.46	0.53	**0.32**	**0.08**
≤ 15	0.86	0.49	0.38	**0.06**	**0.23**	0.39	0.45	**0.25**	**0.03**
≤ 20	0.77	0.49	0.38	**0.06**	**0.18**	0.30	0.21	**0.13**	**0.03**
≤ 30	0.71	0.39	0.38	**0.03**	0.17	**0.07**	0.10	**0.06**	**0.03**
≤ 50	0.41	0.25	0.10	**0.03**	0.05	0.04	**0.03**	**0.03**	**0.02**

Reported values are the mode of k/N for a sample size of 100.
Values in the $\Delta\mu$% column define accuracy bands defined as in Table 5.2. For example: ≤ 10 means maximally 10% accuracy loss starting from the border of the confidence interval for $\Delta\mu = \mu_{\text{spm}} - \mu_{\text{cgspm}}$.

ROC Curves: A ROC curve analysis enables better insights into how link prediction is affected by the coarse-graining. In particular, AUC only captures the average performance of a classifier in ROC space. This evaluation aims to demonstrate that CGSPM is approximating the characteristics of SPM and to reveal a particular weakness of CGSPM: its amplification of classification errors.

Averaged ROC curves for the *jazz*, *USAir*, *email*, and *yeast* graphs are shown in Figure 5.8. The SPM ROC curve is plotted for each graph with a surrounding 95% confidence interval. The interval is based on fitting a binomial distribution to the SPM curve (see Section 2.7.3 for details). The gray lines trace the distribution of the sampled classifiers in ROC space. The best classifiers in each 0.1-sized AUC interval from 0.5 to 1.0 are highlighted with colored lines. Their corresponding relative coarse-grained graph size is reported in the legend of the plots. The full set of ROC plots is shown in Appendix D.5.

The distribution of the classifiers trends towards the shape of the SPM curve. The high AUC curves (red, purple, and brown) start to copy the particular characteristics of the SPM curve. In the *email* plot, a clear TPR jump of the SPM curve just below FPR 0.6 appears in the approximations as they become more accurate. The best classifiers in the *jazz* and *email* plots almost perfectly copy the SPM curve. This trend is seen in all ROC plots and confirms that CGSPM approximates SPM as k approaches N.

Consider the purple ($k = 0.69N$) curve in the *yeast* plot and the brown ($k = 0.63N$) curve in the *USAir* plot. They are both contained within the confidence interval for the SPM curve but they do not copy the SPM curve very closely. These are examples of equivalent performance with different prediction characteristics. The purple ROC curve in the *yeast* plot corresponds to the same classifier where the parameter optimization strategies stop at an AUC maximum as shown in Figure 5.7. The brown curve in the *USAir* plot reaches such an early maximum in AUC space too (see Appendix D, Figure D.3).

Recall that the difference between *yeast* and *USAir* is that CGSPM fails to reach the SPM accuracy level on the former while it surpasses it on the latter. ROC curves enable a better interpretation of this behavior. The SPM curve on the *yeast* graph flattens after it reaches TPR 0.8. However, there is a sudden but significant increase in the TPR after

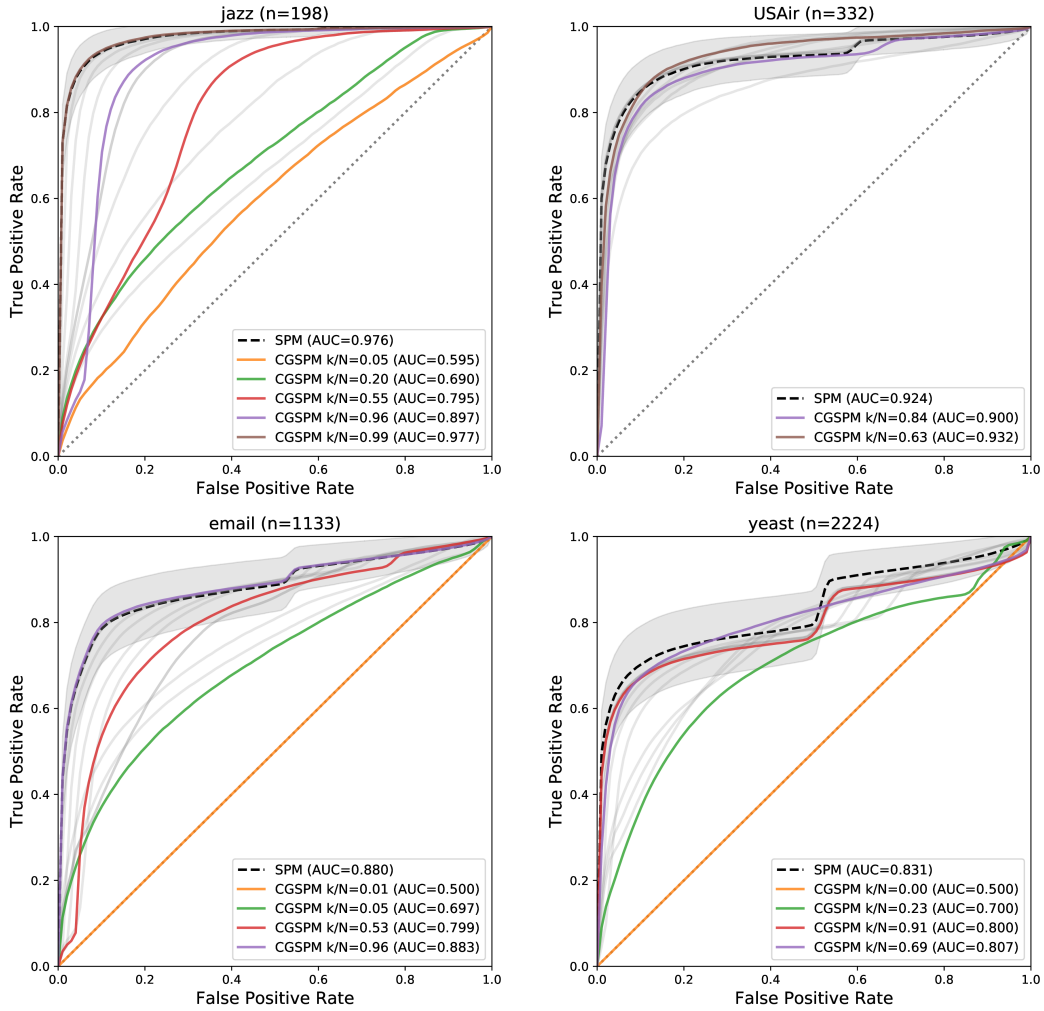


Fig. 5.8: *Averaged ROC curves for a subset of evaluated graphs. Each plot shows CGSPM performance for some of the sampled classifiers (gray curves). The best classifier in each* 0.1*-sized AUC interval from* 0.5 *to* 1.0 *is highlighted with a colored line. The same color indicates the same interval in each plot. Some intervals are empty. The shaded area depicts the 95% confidence interval around the SPM curve.*

the FPR passes 0.5. The CGSPM classifier at $k = 0.69N$ cannot approximate this detail precisely. Instead, the curve is roughly averaged. As a result, approximately half of this TPR increase is missed by the classifier. This explains why it cannot reach the SPM accuracy. In contrast, the ROC curve of SPM on the *USAir* graph flattens at a higher TPR of 0.9. Its TPR has a sudden jump as well but it is much smaller. The corresponding classifier with $k = 0.63$ (brown) fails to copy this feature and flattens out but the jump is less significant and the coarse-grained classifier performs slightly better in this region.

The ROC curves demonstrate another important detail about classification errors. Curves corresponding to smaller k tend to accumulate more FPR before features are copied in ROC space (see purple and red ROC curve for *email*). Furthermore, classification errors

at low TPR are sometimes characterized by a sudden jump in FPR (see purple curve for *jazz* or red curve for *email*). This is a sign that many edges are misclassified at once which is more harmful for AUC than a smooth and gradual accumulation of FPR as seen in the SPM curves.

5.5.7 CGSPM Runtime Results

The Link prediction runtime for all evaluated classifiers are recorded as wall-clock time difference between the beginning of a test until the end of a test. Tests are defined by the two-step process described in the evaluation protocol (Section 5.5.2). Therefore, the measured time represents the time required to compute a score matrix $\langle \hat{\mathbf{A}} \rangle$. All reported runtimes are averages and standard deviation over all tests that share the same classifier model, parameters, and graph. Each CGSPM data point represents the average of $T = 100$ independent runtime measurements. SPM runtimes were averaged from only $T = 50$ samples.

Table 5.6 shows runtime results for CGSPM and SPM. Contours are lines in the parameter space where the relative coarse-grained graph size k/N is constant. For every graph, rows indicate the average runtime measured at the contour lines indicated by the first column. Therefore, columns contain the average runtime at increasing coarse-grained graph sizes for every evaluated graph. The reported runtimes represent the average runtime for any combination of parameters e and b that result in the corresponding coarse-grained graph size. The row labeled "MAX" contains the best measured runtime with maximal coarse-grained graph size. For the same reasons as explained in the accuracy evaluation above, this does not necessarily imply that $k = N$ in the corresponding sample.

The bold values show that at most contours, CGSPM runtime is superior to SPM, even with unrestricted graph size. As mentioned above, this can only be explained by the fact that $k < N$. However, the difference is very small relative to N. On all small graphs, CGSPM has longer runtime in at the maximum contour, indicating, that CGSPM is less efficient than SPM when $k = N$. This is expected because CGSPM does additional work by projecting the matrices to and from the coarse-grained domain.

Next, the relationship of CGSPM runtime and the coarse-grained graph size k is shown for each graph. This aims to provide evidence for the argument that CGSPM allows to exploit beneficial trade-offs not only in terms of accuracy but also in terms of computational cost. Afterwards, the runtime is evaluated as a function in N in order to compare the computational cost of CGSPM and SPM with increasing graph size.

Runtime on each Graph: The columns of Table 5.6 contain the runtime as a function of k/N with fixed N. Figure 5.9 plots two representative examples. Appendix D.7 contains additional plots of all evaluated graphs. Overall, all graphs show very similar patterns.

The *neural* graph plot (left) highlights a patterns seen on small graphs. CGSPM runtime exceeds SPM as k approaches N. However, the optimization strategies both find an AUC maximum at significantly lower runtime. This means that increasing k further has no beneficial effect on link prediction accuracy. This occurs on all except the two smallest graphs in the evaluation. Furthermore, the gap between the curve for $e = 1$ (green) to

Table 5.6: *CGSPM runtime results in seconds at regularly spaced k/N contours*. Bold values show the last contour below SPM runtime (rows at the bottom are better). Columns are ordered ascending by graph size N.*

$\frac{k}{N}$	florida	jazz	neural	USAir	netscience	metabolic	email	hamster	yeast
0.1	0.1±0.03	0.1±0.06	0.1±0.02	0.1±0.02	0.3±0.08	0.3±0.12	0.8±0.04	1.9±0.10	2.8±0.18
0.2	0.1±0.01	0.1±0.02	0.3±0.12	0.3±0.11	0.3±0.05	0.4±0.08	1.4±0.19	3.6±0.20	5.1±0.42
0.3	0.1±0.02	0.4±0.18	0.4±0.15	0.4±0.13	0.4±0.04	0.6±0.08	2.2±0.14	4.7±0.59	7.7±0.54
0.4	0.1±0.02	0.4±0.12	0.5±0.08	0.5±0.07	0.5±0.03	0.7±0.14	3.6±0.26	7.8±0.64	14.0±1.22
0.5	0.3±0.16	0.5±0.12	0.6±0.10	0.7±0.07	0.6±0.04	1.0±0.12	5.0±0.64	12.5±1.12	29.6±4.60
0.6	0.3±0.15	0.5±0.11	0.7±0.08	0.8±0.04	0.9±0.04	1.1±0.12	6.9±0.53	24.4±3.20	58.8±5.17
0.7	0.3±0.14	0.7±0.10	0.9±0.07	1.0±0.03	1.2±0.04	1.6±0.07	9.0±0.68	42.6±3.40	110±5.92
0.8	0.5±0.12	0.7±0.07	1.0±0.05	1.3±0.03	1.6±0.04	2.0±0.07	12.9±0.53	73.5±0.78	175±4.63
0.9	**0.5±0.12**	**0.8±0.05**	**1.2±0.05**	**1.8±0.10**	2.3±0.12	2.5±0.04	22.1±0.54	117±3.35	240±7.94
MAX	0.6±0.04	1.0±0.05	2.1±0.13	2.7±0.20	**3.1±0.21**	**4.1±0.20**	**33.0±0.45**	**125±4.87**	**567±72.79**
SPM	0.6±0.01	0.9±0.03	1.9±0.03	2.3±0.06	3.3±0.03	5.5±0.04	69.6±1.32	300±2.66	1736±11.79

* Contour runtimes are calculated by collecting the last sample mean (of 100 runtimes) in the parameter space located up to and including the contour when scanning along every e dimension. Therefore, each contour is the average of a collection containing up to 10 sample means. Some contours groups contain less samples because the line does not reach large e. All CGSPM values are reported as <mean> ± <standard error> of the contour groups. If the group contains only a single sample, the sample <mean> ± <standard deviation> is reported. SPM values are reported as <mean> ± <standard deviation> for each SPM runtime sample of size 50.

the remaining runtimes is significant. The curves for $e > 1$ (gray) do not show any clear difference.

The *hamster* graph plot (right) is representative of the results observed on large graphs. The gap between $e = 1$ and $e > 1$ disappears and the maximum measured runtime is well below that of SPM, even at $k/N > 0.9$. In theory, it should at least match the SPM runtime when $k = N$. An inspection of the measured k values confirms the largest k values on these graphs are slightly lower than N. As the runtime growth is clearly exponential in k, this small difference between the largest k and N can explain the runtimes differences. The runtimes for the optimized strategies (red and blue) show that the maximal AUC and precision is reached at significantly lower runtimes in most cases.

Runtime at Increasing Graph Sizes: In Figure 5.10, CGSPM and SPM runtimes are plotted as a function in N. The left plot clearly shows exponential growth for SPM as predicted by the computational cost analysis. According to the same analysis, CGSPM runtime grows exponentially as well. The gray lines show that this is plausible, especially as k/N becomes large (steeply rising gray lines). However, there are clearly parameters combinations that offer high accuracy at much lower computational cost. The green curve indicates CGSPM runtime for the best accuracy achieved in the experiments. For any graph with $N > 1000$ it is significantly lower than SPM runtime. The accuracy evaluation has shown that the AUC is at SPM level in these cases except for the *netscience* and *yeast* graphs (see Table C.3).

The right plot in Figure 5.10 shows the same data displayed in a log-log plot. The axes are not equally scaled, therefore, a line indicating quadratic growth ($y = N^2$) is fitted to

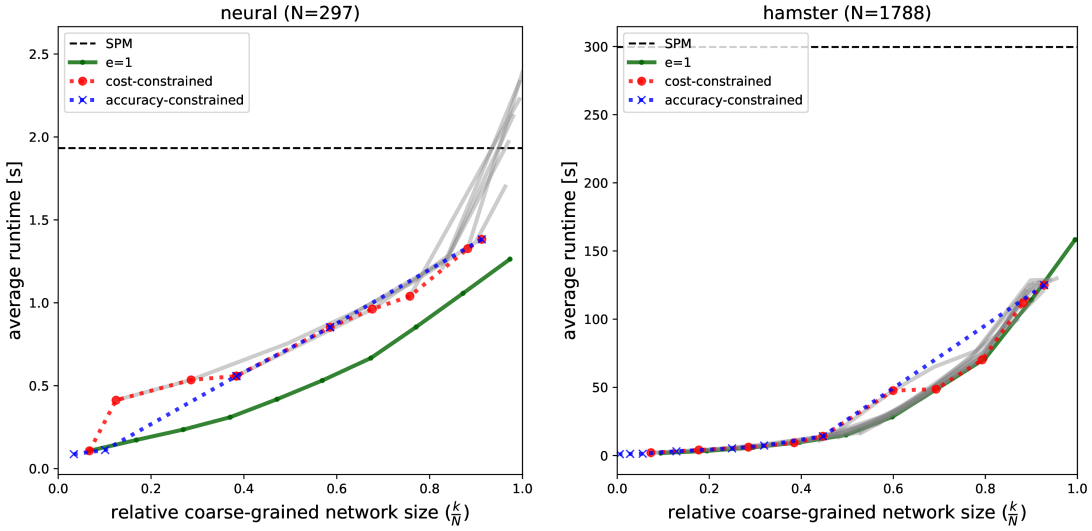


Fig. 5.9: *Average runtime to compute a score matrix* $\langle \hat{\mathbf{A}} \rangle$ *as a function in* k/N *on the* neural *and* hamster *graphs compared to SPM. Dotted curves indicate runtimes with the cost-constrained (red) and accuracy-constrained (blue) optimization strategies. Gray lines show average runtimes sampled from all recorded data.*

the *florida* SPM data point for reference (dark gray dots). As expected, the SPM runtime grows more than quadratic. The CGSPM curves appear to be diverging from the SPM curve which indicates a smaller exponent in their growth function. This fits the complexity analysis as well. However, the evaluated graph sizes are not large enough to make strong conclusions about the time complexity based on the experiments.

5.6 Discussion of Results

As established in the previous experiments, CGSPM approximates SPM even though it does not change the SPM algorithm. Instead, the degree of coarse-graining determines the approximation. Therefore, the coarse-grained graph size k is of special interest. Specifically, k is determined by the choice of the parameters e and b but the relationship of these parameters to k is complex and graph dependent. In the previous section, an empirical evaluation investigated this relationship and has shown how k affects accuracy and runtime. In this section, the observed results are interpreted and related to corresponding theories presented in Chapters 3, 4 and 5. The discussion below follows the structure of the previous section. First, it covers the influence of the parameter and derives heuristics for their choice. Then, the trade-off between link prediction accuracy and coarse-grained graph size is qualified. Finally, the algorithm runtime results and trade-offs are related to the theory about the computational cost that was presented in Sections 3.4.2 and 5.4.2.

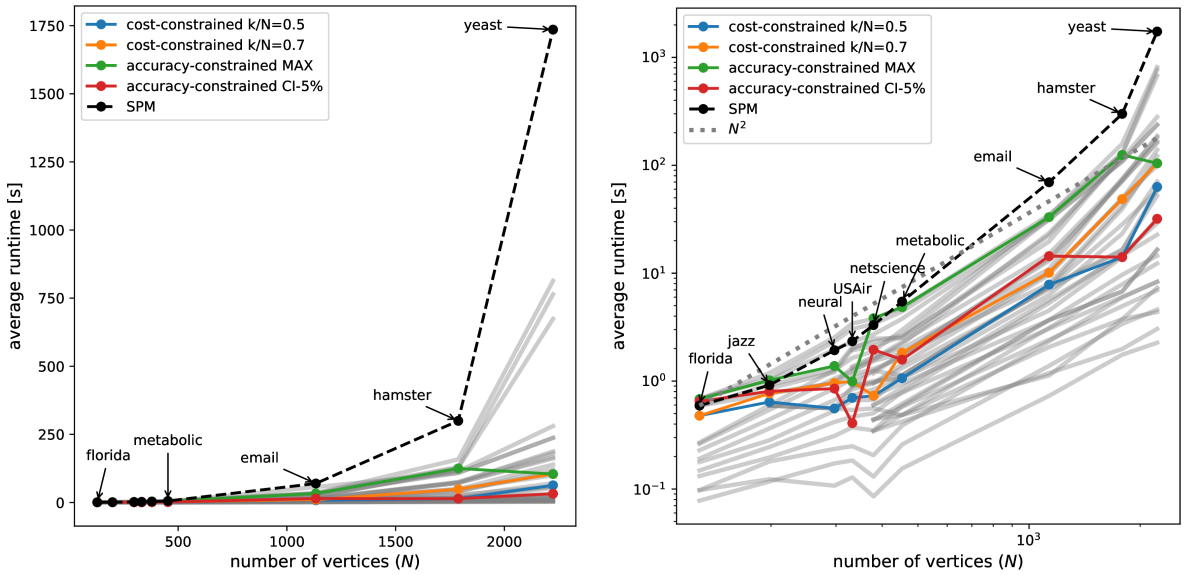


Fig. 5.10: *Average runtime to compute a score matrix $\langle \hat{\mathbf{A}} \rangle$ with CGSPM and SPM for all Graphs in linear (left) and logarithmic (right) scales. For CGSPM, the curves of different cost and accuracy constrained strategies are indicated. Gray lines show average runtimes sampled from all recorded data. The plot on the left shows runtimes with linear axes. The plot on the right shows the same data with logarithmic axes.*

5.6.1 Parameter Choice

Understanding the influence of the parameters e and b on the coarse-grained graph size k is of major importance for the applicability of CGSPM. Parameter choice is complex because CGSPM uses a fixed-size interval based partitioning method. Its advantages and limitations are discussed in Section 4.3. A critical disadvantage of this partitioning method is the lack of control over the size of the resulting coarse-graining, that is, the value of k. The parameter space for any graph is a N dimensional space. An uninformed parameter choice is unlikely to yield reliable results. Fortunately, the experiments demonstrate that interesting trade-offs can be found at relatively small values of e and b. These results allow the formulation of heuristics and strategies for parameter choice.

The parameter space regions where $k > 0.9N$ can be considered uninteresting. Even though the link prediction accuracy is high in those region, any computational cost saving is marginal. Figure 5.4 shows that most of the parameter space appears to be uninteresting. The theory about an explosion of partitions when the segmentation method is becoming constrained by many dimensions and intervals is explained in Section 4.3.4 and predicts this behavior. As a result, the graph cannot be shrunk by a large degree, or at all, when too many eigenvectors are partitioned. On the opposite spectrum, regions with too small k lead to low link prediction accuracy which makes them uninteresting as well.

Good trade-offs between computational cost saving (small k) and high accuracy (large k) are located in a relatively small region distributed along the axes of the parameter

space. All evaluated graphs have shown overlap between regions with high accuracy and significantly reduced k. To choose optimal parameters, an application needs to define a value function. For this purpose, two parameter optimization strategies have been presented and found to provide reliable results. The first chooses parameters that yield the lowest k while maintaining a defined level of accuracy loss. The second method chooses parameters that maximize accuracy constrained by a maximum k, thus limiting computational cost. Parameters chosen with these strategies show analogous patterns on all graphs. Furthermore, optimal parameters are located close to the origin in parameter space, that means, small e and b relative to N. This corresponds to the interesting region described before. Tables C.1 and C.2 list the optimal parameters at different constraints for all evaluated graphs.

Unfortunately, a detailed sampling of the parameter space as conducted for the CGSPM experiments is computationally expensive to obtain. A good heuristic in the absence of this information is to choose e and b close to the origin of the parameter space. A heuristic-based strategy is summarized below.

The main results regarding parameter choice are:

- The parameter space for e and b where interesting values for k can be obtained is limited to small values of both parameters. The center of the parameter space is uninteresting.
- Good parameter locations are particular to each graph and not generally transferable.
- If accuracy scores and the coarse-grained graph size are known or sampled, optimal parameters can be chosen using one optimization strategies presented in Section 5.5.3.
- In absence of detailed information, good heuristics for parameter choice are: Choose $e > 1$ and sample along the values for b. When changing b stops having the desired effect, specifically, when the k becomes too large, then the parameter e should be increased and b reduced. When k becomes sensitive to small changes of a parameter, exploring it further is unlikely to be of benefit.

5.6.2 Link Prediction Accuracy

Link prediction accuracy has been evaluated with two hypotheses in mind. The first hypothesis states that the CGSPM results approximate those of SPM as the coarse-grained graph size k approaches the size of the original graph N. The second hypothesis states that beneficial trade-offs between link prediction accuracy and computational cost can be exploited with CGSPM. These hypotheses are evaluated below. Furthermore, a classification error pattern in CGSPM is discussed.

Approximation of SPM: The first hypothesis compares the link prediction accuracy of SPM and CGSPM. In Section 3.2.2, classical error bounds derived from matrix perturbation theory are used to assess the stability of SPM. Unfortunately these bounds are lose and do not enable strong claims about the accuracy of SPM. Tighter bounds make assumptions about the structure of the graph (or matrix) that cannot be guaranteed and

may even be unlikely in realistic use-cases. Tighter error bounds for SCG have been proposed in related work and improved in Section 4.3.5 of this thesis. However, the SCG error, that is, the error induced by the coarse-graining, and the SPM link prediction error are different errors. Due to the complex interactions between these errors, no theoretical assessment of their relationship is proposed in any related work.

The best characterization of the relationship between coarse-graining error and link prediction accuracy is to construct a relationship between the coarse-grained graph size k to the accuracy of the representation of the original eigenspaces. The primary difference between SPM and CGSPM is the coarse-graining of the input graph. The coarse-graining represents the eigenspaces with low dimensional approximations. Any coarse-graining is defined by the assignment of all vertices to k vertex groups by eigenspace partitioning. When $k < N$, a coarse-graining error is induced because every vertex is projected to the average coordinates of all vertices in the same group. This is explained in Section 4.2.1. As more groups are created, the average distance of vertices to their group center becomes smaller. In the extreme, every vertex occupies a single group exclusively. Then, the error is zero and the eigenspaces can be preserved exactly. However, in this extreme, no size reduction occurs beyond the elimination of indistinguishable vertices. Therefore, the coarse-grained graph size k is a proxy for the similarity of the eigenpairs used by SPM and CGSPM. When $k/N = 1$, the eigenspaces used by CGSPM and SPM are equal and the results should be the same up to stochastic effects.

The relationship outlined above is mirrored in the evaluation results. The increasing similarity between the link prediction characteristics of SPM and CGSPM as k approaches N is verified in the ROC curve analysis (see Appendix D.5). The same behavior is observed for the AUC and precision metrics (see Appendix D.3 and D.4). These results support the hypothesis that CGSPM approximates SPM.

All evaluated graphs that failed to reach the SPM accuracy level also did not reach $k = N$, that is, they always operated under some SCG induced error[7]. This is a limitation of the parameter space exploration and means the required parameter combination for $k = N$ was simply not sampled. As explained before, the sampling is limited to a subset of all possible parameter combinations to make the evaluation feasible. Equivalently, one can say that the structural features that explain the accuracy gap to SPM were not preserved in the coarse-graining. Either the resolution of the eigenvector approximation (parameter b) was too low, or, more likely, some important eigenvectors were not preserved. The latter means that parameter e is too low.

Accuracy Trade-Offs: The second hypothesis relates link prediction accuracy and computational cost with each other. This section discusses the accuracy aspect of this trade-off. The computational cost aspect is analyzed in the runtime evaluation discussion in Section 5.6.3. Both aspects are connected via the coarse-grained graph size k which is a proxy for accuracy and computational cost.

Generally, each application can have a custom definition of a beneficial trade-off. This definition depends on application requirements. In this study, any trade-off with a smaller

[7] On the other hand, many graphs reach the SPM level at $k < N$.

reduction in AUC than in k is considered beneficial. Such trade-offs can be seen where segments of the AUC function have a slope smaller than one. The evaluation results show a consistent pattern of decreasing slope as k approaches N (see Appendix D.3 and D.4). This is also shown and quantified in Table 5.5 which exposes big difference between individual graphs. However, when measuring trade-off in terms of precision, this behavior can not be verified. In high-precision segments, the slope is larger than 1. Therefore, any reduction in k incurs an disproportional loss of precision.

A closer inspection of the differences between individual graphs suggests that the quality of the trade-offs may be related to graph size. This has been predicted by the error bounds derived for SCG in Section 4.3.5. The error bounds suggest that large graphs can be shrunken more than small graphs for the same relative accuracy loss. The evaluation results show such a behavior for the AUC metric. Large differences in this metric can be seen in Table 5.5 between different graphs. Lower values indicate a higher resilience to AUC loss because the graph can be shrunken more while remaining at the same relative AUC level. The two largest graphs are always among the four best values at each accuracy level (marked in bold). However, other factors that have not been isolated may be responsible for some of the differences. Nevertheless, the best half of values (four out of nine) in the table do not contain the smallest graphs. Therefore, there is some evidence that CGSPM may be working better on large graphs. This is an interesting finding when considering that CGSPM is motivated by link prediction problems on large graphs.

In summary, coarse-graining can significantly reduce the graph size while sacrificing only little AUC score on all except one of the evaluated graphs. Therefore, AUC can initially be traded-off at a favorable rate, especially on the larger graphs that have been evaluated. This result supports the accuracy aspect of the beneficial trade-off hypothesis. The same hypothesis cannot be supported for the precision metric. The next paragraphs investigate this discrepancy further.

Classification Errors in CGSPM: In the link prediction context, *true positives* are correctly identified *missing edges* and *false positives* are *non-existing edges* wrongly classified as missing. A classification presumes that there is a *threshold* that determines the mapping of edges to one of these classes depending on whether their score surpasses the threshold. The following study of CGSPM classification errors shows that an application must be willing to accept more false positive classifications to benefit from CGSPM.

CGSPM is shown to have a low precision performance. While the SPM precision level is matched on most graphs in the limit of k/N, it degrades very fast with k and the precision at $k/N < 0.8$ tends to be very low on all graphs. This difference is in stark contrast to the AUC metric. Link prediction precision, as defined by Lü et al. (2015), imposes a particular threshold on the classifier that is set according to the number of edges in the test set. In the experiments, the test set contains approximately 10% of all edges. Any misclassification in the top 10% ranks of the score matrix is detrimental to precision.

The problem with the precision metric is that it assumes each edge to be ranked independently. This assumption is violated by CGSPM because it saves computational

cost by treating groups of similar vertices as one. Therefore, edges in a coarse-grained graph can represent more than a single edge in the original graph. This mechanism is discussed in Section 5.3.1. As a consequence, any false positive classification on a coarse-grained graph can be projected onto multiple edges in the original graph. The number of affected edges is determined by the size of the smaller group connected by that edge. When a misclassified edge connects two large groups, the false positive rate (FPR) increases dramatically and suddenly although only a single classification decision has been made in error. This effect is referred to as *amplification of classification errors* in the remainder of this thesis.

Large groups that can cause big amplifications are more likely to occur when the coarse-grained graph size is small. In that case, the risk of a false classification increases because much information is removed from the graph. Suppose, a significant coarse-graining shrinks the graph size by 50% or more. However, the classification threshold is not adapted to account for this fact. In sparse graphs, $O(N)$ edges are assumed. A very naive extrapolation suggests that each coarse-grained edge connects groups of average size two in this scenario. Then, the effect of any false classification is doubled on average. This is likely the most relevant reason for the bad performance in terms of precision.

AUC scores are affected less by error amplification because there is no reliance on a particular threshold. AUC quantifies the probability that randomly chosen missing edges are ranked better than randomly chosen non-existing edges. This gives an assessment of a classifier's ability to rank positives at the top and negatives as the bottom without imposing a specific threshold.

The ROC curves in Figure D.5 show a behavior that can be explained by error amplifications. Sudden jumps in FPR indicate that a large group of edges is misclassified. Additionally, there is a pattern of jumps in true positive rate (TPR) of the SPM curve. They are copied very precisely by some CGSPM classifiers; but only after a larger amount of FPR has been accumulated. This suggests that classification errors have been amplified while the classifier characteristics remain very similar. Even though AUC is less affected, the amplification of classification errors can become very detrimental to the AUC scores as k becomes small.

ROC plots can be used to determine optimal classification thresholds. Assuming true and false positives are considered equally important, a good location in ROC space is where the curves come closest to point (0,1) after accumulating most of the TPR. As curves flatten and go towards point (1,1), the classification trade-off is unfavorable as any increase in TPR accumulates a disproportional amount of FPR. Refer to Section 2.7.1 for a detailed discussion of classifiers in ROC space.

Summary: The evaluation verifies several of the claims made about the relationship between k and link prediction accuracy. Furthermore, the amplification of classification errors by CGSPM has been analyzed in some detail. The conclusion about the link prediction accuracy of CGSPM are:

- The AUC and precision metrics measured with CGSPM approach the SPM levels as k approaches N on all graphs.
- When $k \approx N$, CGSPM accuracy is approximately equal the SPM accuracy in terms of AUC and precision on all but two evaluated graphs.
- The shapes of the AUC functions show that beneficial accuracy trade-offs are possible on most graphs and AUC can be preserved even when the coarse-graining is significant.
- CGSPM amplifies classification errors. Therefore, the precision metric cannot be preserved at significant coarse-graining levels and any application that wants to use CGSPM must accept a higher false positive rate to be able to exploit trade-offs.

5.6.3 Link Prediction Runtime

Computational cost is related to algorithm runtime but there are many confounding factors. Some are discussed in Section 5.4.3. Despite these distortions, clear patterns emerge in the experiments which can be explained by the computational cost theory and complexity analysis of SPM and CGSPM. The aim of the following discussion is two-fold. First, it is shown that the runtime results confirm the beneficial trade-off hypothesis for the computational cost aspect. Second, the evidence supporting the time complexity analysis of SPM and CGSPM (see Sections 3.4.3 and 5.4.3) is discussed and extrapolated to larger graph sizes.

The analysis in Section 5.4.2 established the computational cost of CGSPM to be $(2ks + s + 1)N^2 + O(k^2N)$ floating point operations (Eq. 72). For any given graph and application, N and s are constant. Therefore, the computational cost is expected to exhibit at least quadratic growth in k. In this limit, the cost is estimated at $(2s+1)N^3 + O(N^2)$ operations which is similar to that of SPM which has been established at $(3+\eta)sN^3 + O(N^2)$ floating point operations in Section 3.4.2, Equation (49).

The runtime evaluation in Section 5.5.7 measures average algorithm runtime instead of floating point operations. Due to multiple confounding factors, conclusions made from computational cost analysis cannot be directly transferred to runtime. However, a relationship clearly exists and the dominant tendencies in computational cost should be reflected in algorithm runtime.

The expected exponential relationship in k is seen in the runtime evaluation results, for example, in the plots presented in Appendix D.7. As k approaches N, the CGSPM runtime grows exponentially. On large graphs, CGSPM is seen to be faster even when $k \approx N$. For any regime where $k < N$, CGSPM is clearly faster than SPM on all graphs. These results show that the coarse-graining approach reduces computational cost significantly. As established before, good accuracy trade-offs are possible at $k/N < 0.5$ (e.g., Table 5.5). In the corresponding segments of the runtime function, the trade-off is decisively beneficial in terms of runtime. Therefore, the beneficial trade-off hypothesis can be fully confirmed on the evaluated graphs.

It is more difficult to assess the time complexity of CGSPM based on the evaluation results because the evaluated graphs are too small to fully exhibit their limiting behavior. Nevertheless, some insights can be gained about the relationship of k and N by extrapolation of the estimated runtimes.

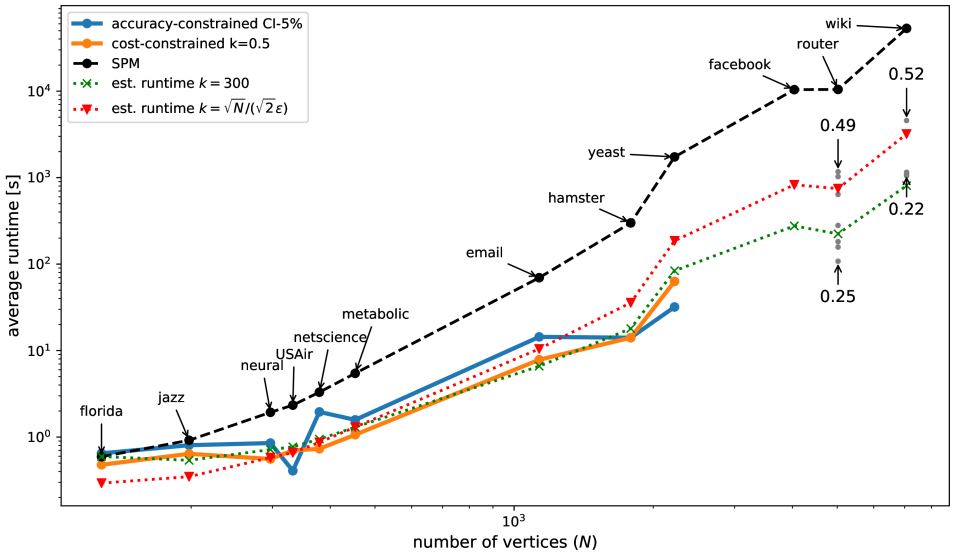


Fig. 5.11: *Estimated CGSPM runtimes in a log-log plot. The green dotted line shows the computation time estimated with Equation* (73) *and* $k = 300$ *(independent of* N*). The red dotted line shows the same cost estimate but assuming* k *and* N *relate as described by Equation* (65) *(with error* $\epsilon = 0.05$*). Additional runtime results for three larger graphs are indicated but optimal parameters are unknown in these regions.*

Let $s = 10$ and $\eta \approx 2.6$ due to the assumptions made in Section 3.4.3. Substituting these values in Equations (49) and (72) (shown above). The relative difference in their computational cost is assumed to also govern the difference in runtime. Therefore, the runtime of CGSPM can be estimated as

$$\theta_{\text{CGSPM}} = \frac{\theta_{\text{SPM}}\ (2ks + s + 1)N + k^2}{(3 + \eta)sN^2 + N} \approx \frac{\theta_{\text{SPM}}\ (20k + 11)N + k^2}{56N^2 + N}, \tag{73}$$

where the CGSPM runtime is denoted θ_{CGSPM} and θ_{SPM} represents a runtime measurement of CGSPM on the same graph.

The critical question to answer in regard of the time complexity is whether k can be assumed independent of N. The best theory available as to how k and N relate is given by the SCG error bounds. Specifically, Equation (65) describes the relationship for both quantities when the SCG error is constant. Figure 5.11 plots the runtimes of SPM and CGSPM with two parameter optimization strategies. The blue curve plots the runtime constrained to maximally 5% accuracy loss and the orange line is the runtime constrained by $k \leq 0.5N$. Furthermore, the estimated CGSPM runtime defined by Equation (73) is plotted with the assumption of k independent of N (green) and the alternative hypothesis (red) of N and k relating as in Equation (65). Note, the axes are both logarithmic and not scaled equally.

The former assumption requires to fix a value for k and the latter depends on the SCG error ϵ. The values were chosen empirically as follows. Inserting N and k of all reported evaluation results into Equation (65) suggests that $\epsilon = 0.05$ is a (loose) lower bound

for high accuracy results (CI-10%). Inserting the evaluated graph sizes and $\epsilon = 0.05$ into Equation (65) results in values for k that are in the order of magnitude 10^2. Then, $k = 300$ was chosen arbitrarily. Changing this value within the same order of magnitude does not affect the plotted slope significantly.

Further results were computed for the following graphs of larger size: *facebook* ($N = 4039$), *router* ($N = 5022$), and *wiki* ($N = 7066$). Due to their size, fewer samples were computed for these graphs. The SPM runtimes of *facebook*, *router*, and *wiki* are averaged from 50, 30, and 15 samples respectively. Furthermore, optimal CGSPM results are not available for these graphs because the parameter space was only sampled heuristically. The corresponding runtimes are indicated as gray dots in Figure 5.11, the shortest and longest runtime for each graph is annotated with the corresponding k/N value.

Where data is available, both cost estimates appear to be reasonably accurate considering all confounding factors. However, a decision whether the independent k estimate (green) or the dependent k estimate (red) is more accurate is not possible based on the results. On the other hand, the gaps between both CGSPM estimates and SPM are clearly widening. If either estimate is accurate, the runtime growth of CGSPM is below that of SPM. This observation supports the time complexity analysis that predicts a lower complexity for CGSPM.

The main results from the runtime evaluation can therefore be summarized as:

- CGSPM outperforms SPM in terms of runtime for any significant coarse-graining. Furthermore, the shape of the runtime function confirms that beneficial trade-offs in terms of runtime can be found in all evaluated graphs.
- Based on the evaluation results, the relationship between k and N cannot be determined with certainty.
- The measured CGSPM runtime grows with a smaller exponent than that of SPM. A superior efficiency can be confirmed which supports the time complexity analysis.

5.7 Limitations

This sections discusses a selection of topics that limit the validity of the presented results or the capabilities of the CGSPM method. Furthermore, CGSPM and a previously used random graph sampling method are compared.

5.7.1 Evaluation Result Limitations

The CGSPM evaluation is limited to graph sizes that can be considered relatively small. The largest evaluated graph is *yeast* with $N = 2224$. This has two primary reasons. First, the related work is evaluated on the same graphs and CGSPM is compared against it. Second, the entire evaluation depends on optimal parameters found by a grid search of the parameter space. An exploration of equal quality for larger graphs is too time consuming.

Nevertheless, the evaluation could answer most research questions and hypotheses except for the exact relationship between k and N. The evaluated graph sizes are not

large enough to test the limiting behavior of the method. A few runtimes were measured for larger graphs up to $N = 7066$ and compared with extrapolated, estimated runtimes in Figure 5.11. However, nothing is known about the optimality of the used parameters. Therefore, the best complexity assessment of CGSPM remains the theoretical analysis.

The implemented and evaluated algorithms are all sequential although they can be parallelized with relative ease. Parallelization was briefly discussed in Sections 3.4.1 and 5.4.1 for SPM and CGSPM respectively. With parallel algorithms the observed runtimes should be significantly better on large graphs.

5.7.2 CGSPM Method Limitations

The conclusion about the optimal parameter value for e to be small has to be put in context of the eigenspace partitioning method used in CGSPM. The reason why large e does not lead to optimal results is not because the preservation of many eigenpairs leads to bad accuracy but because the partitioning method is unable to significantly shrink the graph in this case. In fact, the reason why the SPM accuracy level is not reached on some graphs, for example on *yeast*, may be due to a low number of preserved eigenpairs.

CGSPM has been defined to preserve the leading eigenpairs of a graph via parameter e. However, adjacency matrices can have negative eigenpairs whose absolute value is large. CGSPM ignores these eigenpairs which can cause significant error. An extension to include these eigenpairs is relatively straight forward because the iterative methods used for the partial eigendecomposition can locate eigenpairs at both ends of the spectrum. However, it has not been implemented in this work.

A similar limitation is given by the definition of parameter b. The parameter value is the same for each eigenpair. However, a distinct advantage of the segmentation method is that each eigenpair can be approximated with custom precision. CGSPM does not allow to use different b values. An extension to provide a different b value for each eigenpair is simple to implement but such an extensions increases the complexity of parameter choice which is already a concern with CGSPM.

It should be stressed that CGSPM loses information primarily about edges connecting vertices inside the same partition. Vertices that are grouped together are characterized by having similar sets of neighbors. In social network contexts, such groups are sometimes referred to as communities or neighborhoods. CGSPM can predict all types of edges but two effects need to be distinguished.

1. Local effect: Similar vertices become indistinguishable in SCG. When vertices are assigned to a common partition, their interactions are averaged. Coarse-grained vertices have self loops that contain aggregated information about the connectivity inside of a partition but their structure is not represented in the coarse-grained graph. After reverse projection, all vertices that shared a partition have some degree of mutual interactions in the score matrix. The fact that they are grouped together is adding to link-existence estimates between vertices in the same partition.
2. Global effect: SPM predicts edges between different partitions based on dominant patterns in the structural features of the graph. Edges connecting different partitions

or communities can be predicted based on the structural information preserved in the coarse-graining. In other contexts, such edges are sometimes called long-range links.

As a consequence, inter-partition edges are likely predicted at better accuracy because their structure is represented in coarse-grained graphs. Intra-partition edges have no detailed representation and their ranking is determined by the aggregated connectivity of their neighborhood. To increase accuracy for link prediction between vertices that are grouped together, additional steps should be considered in the processing pipeline.

5.7.3 Difference to Sampling

A common method for the circumvention of computational bottlenecks in graph algorithms is to sample a subset of edges of a large graph and to verify an algorithm's performance on multiple independent samples. Leskovec and Faloutsos (2006) demonstrated that a randomly sampled subgraph that represents only 25% of the full graph has its singular values and vectors scaled proportionally to the sampling factor. Lü et al. (2015) have validated their structural consistency index on sampled subgraphs as well. A sample consisting of about 50% of the edges approximates the structural consistency of the full graph with good precision for most of the evaluated graphs. However, as demonstrated in the CGSPM evaluation, computing SPM on the full graph instead of a sampled graph achieves a higher accuracy. Some results reported in Lü et al. (2015) have been obtained on graph samples. In Figure 5.12, the SPM AUC values computed on the original graph are compared to those reported by Lü et al. (2015) for sampled subgraphs. The accuracy of the former is consistently and significantly better. Therefore, the accuracy of CGSPM can be higher than that reported in Lü et al. (2015) for SPM.

There is also a semantic difference between a coarse-graining approach and graph sampling. Only edges connecting vertices that exists in a sample can be predicted by the latter. There is no general way to transfer its predictions to edges that were not included. In contrast, CGSPM always predicts scores for all edges in a graph.

Both methods can be useful in different situations. Sampling is likely more localized and can better capture short-range patterns in a graph but when the sample is too small it can miss some global patterns entirely while CGSPM is a global link prediction method and, due to the coarse-graining effect described before, it predicts structurally dominant long-range edges better than local structures.

5.8 Conclusion

The CGSPM link prediction method represents a new approach to approximate SPM (Lü et al., 2015). SPM has been shown to be a powerful link prediction method but its complexity makes it infeasible to scale up to large graphs. CGSPM uses the spectral coarse-graining framework (de Lachapelle et al., 2008) to shrink and approximate the input graph. An application can then tune two parameters to find approximations that provide a beneficial trade-off between link prediction accuracy and computational cost. The main contributions presented in this chapter are the CGSPM method definition and an extensive evaluation of the approach.

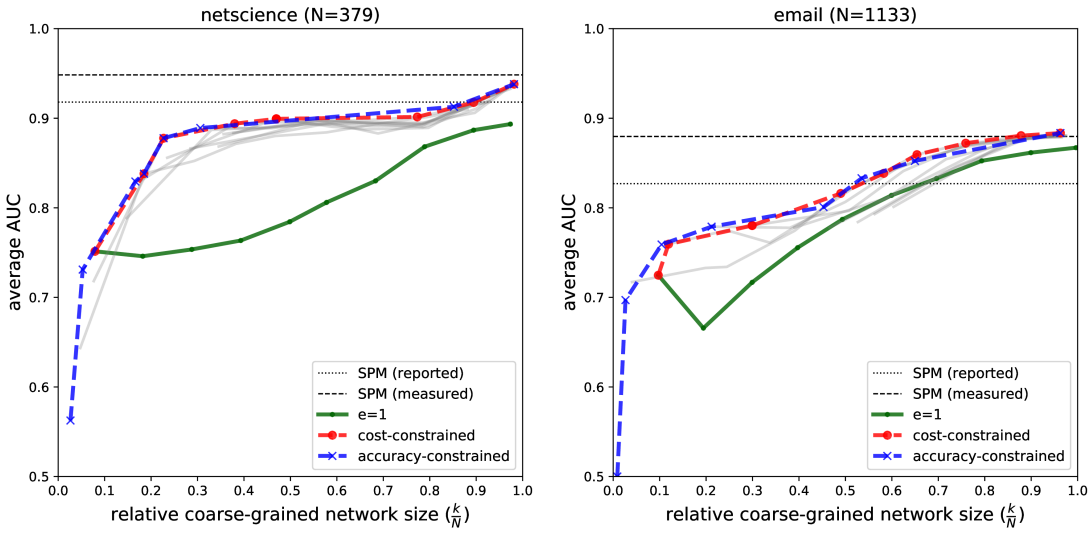


Fig. 5.12: *Measured versus reported AUC. The dashed SPM line is the measured value on the full input graph. The finely dotted SPM line shows the AUC reported in Lü et al. (2015). The reported line is obtained on a randomly sampled sub-graph of* 300 *vertices. More examples are shown in Figure D.3. The corresponding precision results are shown in Figure D.4.*

The influence of each parameter on the size of the graph approximation and on the link prediction accuracy is demonstrated. Its evaluation reveals a non-trivial relationship between parameter choice and the resulting graph approximation. Applications able to afford a parameter search can use two presented optimization strategies that find beneficial trade-offs between accuracy and computational cost. Otherwise, applications can use heuristics that have been established based on the evaluation results.

The link prediction capabilities of CGSPM have been shown to be equal to those of SPM. Additionally, CGSPM offers the ability to reduce computational cost at the expense of accuracy. An extensive evaluation of this trade-off determined that accuracy loss manifests primarily in an increased false positive classification rate. For applications that can afford a higher number of false positives, beneficial trade-offs exist and CGSPM can exploit them.

A theoretical evaluation of the computational cost of CGSPM relates the number of operations required to compute link-prediction results to the coarse-grained graph size k. Based on the operation count, the time complexity of CGSPM is shown to be $O(kN^2)$ where N is the input graph size. When k is small and independent of N, this represents a considerable improvement over SPM. The latter has a complexity of $O(N^3)$. The exact relationship between N and k varies depending on parameter choice. This is a consequence of the inherent trade-off between computational cost and accuracy. Newly derived error bounds for spectral coarse-graining suggest that for practical purposes, accuracy loss can be limited while operating at a complexity of $O(\sqrt{N}N^2)$. The error bounds also imply that the trade-offs are better for large graphs than small ones. The resulting computational

cost reduction has been shown to be significant which confirms the viability of the coarse-grained link-prediction approach.

Furthermore, the presented coarse-graining framework can be generalized or extended. It is mostly agnostic to SPM and produces a graph approximation with the same characteristics as the original graph. Any application that takes the original graph as input can, in principle, work on a coarse-grained graph. The existence of beneficial trade-offs is likely application-specific and needs to be verified in each case. Nevertheless, this approach offers a potential solution for further use cases or combinations of applications. One can imagine to pre-process or post-process the predicted graph in the coarse-grained domain, for example, to produce visualizations.

To summarize, CGSPM offers a solution for SPM link-prediction use-cases that aim to scale-up to larger graphs or to speed-up existing tractable computations. The complexity is reduced in theory but depends on parameter choice. CGSPM is most useful when parameters can be re-used, that is, in use-cases requiring repeated computations on graphs of equivalent structure.

Chapter 6

Efficient and Controllable Coarse-Graining

The coarse-grained structural link prediction method (CGSPM) proposed in Chapter 5 enables applications to exploit trade-offs between computational cost and link prediction accuracy. A good choice of parameters leads to link prediction with superior runtime performance and close to maximal accuracy. However, the choice of optimal parameters for CGSPM is a complex problem in itself. This chapter proposes an extension to CGSPM that simplifies the choice of parameters and improves link prediction accuracy further.

6.1 Introduction

In the previous chapter, a coarse-grained SPM method (CGSPM) was presented. It has shown that the link prediction results of the original SPM method by Lü et al. (2015) can be approximated with coarse-grained input graphs. Coarse-graining reduces the input graph size and this lowers the computational cost of the algorithm. CGSPM was evaluated on graphs of different sizes and has been able to exploit beneficial trade-off between accuracy reduction and algorithm runtime.

While this approach could be validated in the the previous chapter, a number of limitations have been discovered as well. In particular, a design decision to employ the so-called segmentation method for eigenspace partitioning (see Section 4.3.4) is problematic. This method is attractive for its simplicity and computational efficiency but it makes the coarse-grained graph size difficult to control. The evaluation of the parameter space has shown that this lack of control severely limits the ability of CGSPM to gain useful trade-offs from most of the parameter space. Furthermore, it is very hard to choose optimal parameters.

In this chapter, a modification is proposed to make parameter choice more flexible and predictable. This is achieved with a different eigenspace partitioning method. Additionally, the efficiency of the eigenspace approximation is increased which allows to preserve significantly more eigenspaces. This can be used to improve link prediction accuracy further. The proposed extensions is called efficient and controllable structural perturbation method for link prediction (ECSPM). It changes only the spectral coarse-graining without modifying the remaining link prediction process.

In CGSPM, the eigenspace partitioning and link prediction process can be summarized as follows. There are two parameters, e and b. The parameter e selects eigenspaces for partitioning and b determines the approximation accuracy. Each selected eigenspace is first partitioned separately and then, all partitionings are combined with the segmentation method as described in Section 4.3.4. The resulting coarse-graining projector maps the input graph onto a subspace that can be represented with a smaller, coarse-grained graph of size k. All link prediction results are computed on this coarse-grained graph. The

partitioning method has linear complexity but does not allow control over the coarse-grained graph size as it is determined in an unpredictable and graph-dependent manner.

The first problem addressed by ECSPM is the lack of control over the coarse-grained graph size. When the effect of parameter choice cannot be reliably predicted, it is difficult to sample good parameters. Any sampling strategy must rely on imprecise heuristics which likely requires many samples to find good parameters. Parameter sampling is computationally very expensive and once established parameters can not be transferred to different graphs as the location of optimal parameters is graph dependent.

ECSPM changes the eigenspace partitioning method to a process called k-means clustering. K-means directly takes the coarse-grained graph size k as a parameter instead of the problematic parameters e and b which determine k in an unpredictable manner. Therefore, the coarse-grained graph size is fully controllable and all theoretical knowledge and experimental evidence about k from the previous chapters can be used to estimate the link-prediction trade-offs and time complexity. The need to conduct an expensive parameter space search is reduced. The properties of k-means eigenspace partitioning are extensively discussed in Sections 4.3.3 and 4.3.4.

The second problem addressed by ECSPM is the high computational cost of obtaining the required eigenspaces, that is, the leading eigenvectors of the graph matrix. In CGSPM, e is limited to very small numbers. A large e increases the computational cost significantly because e is the number of eigenpairs that need to be computed. As e approaches k, the complexity of a partial eigendecomposition becomes $O(k^2N)$ for sparse graphs (see Section 2.4). Depending on the relationship between k and N, this cost can be significant and increase the overall algorithm complexity. Furthermore, a large value for e restricted the previously used partitioning method and no significant size reduction of the coarse-grained graph could be achieved. When $k \approx N$, any advantage from using the CGSPM approach is lost.

ECSPM introduces a significantly more efficient spectral coarse-graining method that avoids the eigendecomposition. Since this is computationally much more efficient, a larger number of eigenspaces can be preserved in the coarse-graining while maintaining computational efficiency gains. With the k-means partitioning method larger values for e can be chosen without restricting the size of the coarse-graining. In fact, ECSPM preserved the maximal number of eigenspaces for a coarse-graining, that is $e = k$. These advantageous features are enabled with polynomial approximation methods and random projections. With these methods sparse matrices can be approximated with complexity $O(ebN)$, where b is the order of the polynomial approximation. Choosing $e = k$ is computationally feasible and k can be chosen independently of N.

The main contributions presented in this chapter are:

- The definition of an extension to CGSPM with more flexible and predictable parameter choice.
- A significantly more efficient SCG partitioning method for sparse graphs which is based on Chebyshev polynomial expansion filtering.
- A detailed computational cost analysis of the polynomial approximation approach.

In summary, ECSPM uses k-means clustering for eigenspace partitioning which greatly simplifies the parameter choice by making the size of the coarse-grained graph controllable. The circumvention of the eigendecomposition reduces the computational cost of the coarse-graining. This efficiency gain is exploited for a more precise coarse-graining which yields better accuracy at smaller coarse-grained graph sizes.

The remainder of this chapter is organized as follows. Section 6.2 presents a review of related work. The ECSPM eigenspace partitioning method is defined in Section 6.3. The ECSPM algorithms, a discussion of the employed approximation methods, and a detailed analysis are presented Section 6.4. In Section 6.5 an experimental evaluation of ECSPM is conducted. Its results are analyzed in detail in Section 6.6. This chapter concludes with a summary of contributions and limitations in Section 6.7.

6.2 Related Work

Polynomial expansion filtering methods have been developed in different fields and disciplines. The following review of related contributions focuses on important milestones that introduce methods used in ECSPM, that is, the efficient approximation of eigenspace subsets or spectral embeddings with polynomial expansion filtering and combinations with random projections.

An early contribution that used Chebyshev polynomials on matrices has been proposed in Greenbaum and Trefethen (1994). It uses matrix polynomials for the approximation of Arnoldi and Lanczos iterations. The proposed approach avoids the materialization of a potentially very large matrix, an optimization that is also used in ECSPM. In Toh and Trefethen (1998), Chebyshev polynomials of matrices are defined more generally and the authors attribute the origins of this approach to Faber (1920).

In computational physics, a divide and conquer method to solve the Kohn-Sham equation has been proposed by Schofield et al. (2012). This method essentially solves a large eigenvalue problem by "spectrum slicing", that is, a polynomial expansion filtering of a large hermitian matrix to isolate different eigenspaces. Each eigenspace-interval (slice) can then be approximated in isolation with the Rayleigh-Ritz method which is an alternative to the Arnoldi/Lanczos iteration. Schofield et al. also use Jackson smoothing to correct for the Gibbs phenomenon (see Section 2.6.4). This optimization appears to have been first used by Silver et al. (1996) for this purpose. Di Napoli et al. (2016) use the spectrum slicing approach to efficiently count eigenvalues of matrices in an arbitrary interval. The counting is based on a variant of Hutchinson's stochastic estimator (Hutchinson, 1990) which bears strong similarities to random projections.

Hammond et al. (2011) translated methods from the field of signal processing to the study of complex graphs. This work is situated in the field of graph signal processing and works with the Laplacian graph matrix. Its spectrum is interpreted as graph frequencies which can be filtered with polynomial expansion filtering. ECSPM takes a very similar approach but uses the adjacency matrix instead.

In a separate line of work, Ramasamy and Madhow (2015) combine polynomial expansion filtering approaches on general, non-square matrices with random projections.

Their work computes approximate *spectral embeddings*, that is, the projection of vectors onto a spectral domain without the need to compute a singular value decomposition (SVD). Spectral embedding is defined in various flavors, for example, principal component analysis (PCA) can be interpreted as a spectral embedding. The authors propose to approximate matrix functions with Legendre polynomials which belong to the same family as the Chebyshev polynomials used in ECSPM.

At the same time, Tremblay et al. (2016a) used polynomial expansion filtering and random projections on Laplacian graph matrices to propose an accelerated spectral clustering algorithm expressed in the graph signal processing framework. In Tremblay et al. (2016b) their method is extended to approximate arbitrary interior eigenvalues of the Laplacian spectrum based on the eigenvalue count method of Di Napoli et al. (2016). These eigenvalue estimates allow the efficient definition polynomial filter functions. Furthermore, Jackson smoothing is adopted as defined by Schofield et al. (2012).

Finally, Paratte and Martin (2016) approximate eigensubspaces of the Laplacian matrix with polynomial expansion filtering and random projections based on the work of Tremblay et al. (2016b). Their approach can be interpreted as mostly equivalent to the method proposed by Ramasamy and Madhow (2015). However, the final subspace approximation uses an additional singular value decomposition of the low-dimensional eigenspace embedding in order to obtain a full set of orthonormal singular vectors. Furthermore, this work improves the eigenvalue estimation algorithm proposed in Tremblay et al. (2016b).

Note, Tremblay et al. (2016b), Paratte and Martin (2016), and Ramasamy and Madhow (2015) use almost the same combination of methods in different contexts. In all cases, a distance-preserving spectral embedding is obtained from a random projection. ECSPM employs the same methods to compute spectral embeddings of vertices and uses these embeddings as feature vectors for eigenspace partitioning and spectral coarse-graining.

6.3 Method

ECSPM changes the first step of the coarse-grained link prediction process, that is, the coarse-graining of matrix $\mathbf{A}$ to obtain a coarse-grained matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{k \times k}$ where $k \leq N$. The most important part of this process is the partitioning of all graph vertices into groups such that a subset of eigenpairs is preserved in the final coarse-graining. Graph vertices are embedded in a partitioning space at coordinates given by the rows of the eigenvector matrix $\mathbf{U}$. The relationship that associates each vertex to the a row of $\mathbf{U}$ is defined in Section 2.3.4. These row-vectors are points in the subspace spanned by the eigenvectors and they can be partitioned using different clustering algorithms.

Before defining the changes made by ECSPM, some definitions are reiterated. In CGSPM, computational cost is reduced by using only a subset of e columns of $\mathbf{U}$. The parameter e is a strictly positive integer that defines the number of leading eigenpairs to preserve in the coarse-grained matrix. Any choice of $0 < e \leq k \leq N$ induces a corresponding eigenvector matrix $\mathbf{U}_e = [\ u_1\ \cdots\ u_e\ ]$. The rows of $\mathbf{U}_e$ are e-dimensional vectors and used to embed each of the N vertices of V into a partitioning space that is then clustered into k groups.

To circumvent the eigendecomposition, ECSPM requires an equivalent set of vectors. Suppose such vectors are given by the rows of a matrix $\mathbf{M} \in \mathbb{R}^{N \times e}$. In the following subsections, the matrices $\mathbf{U}$ and $\mathbf{M}$ are related trough a spectral embedding framework. Thereafter, a matrix function is defined that preserves the e leading eigenpairs exactly and random projections are employed to embed the matrix function in an e-dimensional subspace; this embedding is given by the rows of matrix $\mathbf{M}$ and it replaces the rows of $\mathbf{U}_e$ in the SCG partitioning. The described framework is broadly equivalent to the approach proposed in Ramasamy and Madhow (2015), although adapted to the case of real symmetric matrices and the computations are implemented with different methods.

The last subsection summarizes the combination of methods used in ECSPM and connects them with the remaining link prediction process. All computational aspects, including the polynomial filtering approximation of $\mathbf{M}$, are discussed in Section 6.4.

6.3.1 Spectral Embedding Framework

Consider an univariate function $h : \mathbb{R}^{N \times N} \to \mathbb{R}^{N \times N}$ where the variable is a matrix. This is called a matrix function and it can be defined as

$$h(\mathbf{A}) \coloneqq \sum_{i=1}^{N} h(\lambda_i) u_i u_i^\top .$$

This definition should remind the reader of the eigendecomposition as defined in Equation (3). Up to a scalar function $h : \mathbb{R} \to \mathbb{R}$ defined on the eigenvalues of $\mathbf{A}$, the formulas are equal. Matrix functions are discussed in more detail in Section 6.3.2. For now, it is sufficient to see that defining the scalar function as $h(y) \coloneqq y$ results in matrix $\mathbf{A}$. Therefore, any matrix for which an eigendecomposition exists can be expressed in this manner and matrix functions and matrices can be used interchangeably.

Suppose the rows of $h(\mathbf{A})$ are projected onto the eigenbasis, $h(\mathbf{A})\mathbf{U}$. This is analogous to the projection that defines the association of vertices and eigenvector components in Section 2.3.4. Using Equation (10), the spectral embedding of all "vertices of $h(\mathbf{A})$" in the spectral domain of $\mathbf{A}$ is

$$\widehat{\mathbf{V}} = [\ h(\lambda_1)u_1\ h(\lambda_2)u_2\ \cdots\ h(\lambda_N)u_N\ ]. \tag{74}$$

The only difference between the components of each column of $\widehat{\mathbf{V}}$ and those of $\mathbf{U}$ is that the former are scaled by a factor $h(\lambda_i)$ which is constant for each column. This is analogous to Section 2.3.4, where each vertex embedding has been found to be scaled by a constant factor λ_i.

When $h(\lambda_i) \neq 0$, the result of the SCG partitioning of $\widehat{\mathbf{V}}$ with any of the methods described in Section 4.3 is equivalent to a partitioning of $\mathbf{U}$ with the same method. To see this, recall that SCG minimizes Equation (58); a cost function based on the euclidean distance of an error vector determined by the distance of each embedded vector to the barycenter of the group it is assigned to. Uniform scaling of each coordinate does not change the relative error nor the barycenter of any group. Therefore, a deterministic optimization process using only relative pairwise distances converges on the same optima.

Suppose each row of $\widehat{\mathbf{V}}$ is projected back onto the standard basis of $\mathbb{R}^N$. This projection is $\widehat{\mathbf{V}}\mathbf{U}^\top$. Because $\mathbf{U}$ is an orthogonal matrix, $\mathbf{U}^\top\mathbf{U} = \mathbf{I} = \mathbf{U}\mathbf{U}^\top$. As a consequence, the projection is only a rotation (change of basis) and recovers $h(\mathbf{A})$ exactly as seen in the following equation,

$$\widehat{\mathbf{V}}\mathbf{U}^\top = h(\mathbf{A})\mathbf{U}\mathbf{U}^\top = h(\mathbf{A}).$$

Rotations and other orthogonal transformations preserve euclidean distance exactly and only change the coordinate system in which vectors are expressed. This is shown in Section 2.3.3. It can be concluded that the rows of $\widehat{\mathbf{V}}$ have the same pairwise distances as the rows of $h(\mathbf{A})$ and either matrix can be used for eigenspace partitioning.

At this point it is important to remember why the eigendecomposition is used because above derivation suggests that partitioning $\mathbf{A}$ and $\mathbf{U}$ leads to the same result. However, the eigendecomposition is very convenient for dimensionality reduction. When all eigenpairs are known, a set of columns associated to small eigenvalues can simply be removed from the eigenvector matrix. Removing eigenvectors associated to small eigenvalues has little effect on the interaction described by $\mathbf{A}$. However, this does not work when the eigenpairs are unknown because the rows (and columns) of $\mathbf{A}$ are linear combinations of unknown eigenpairs whose contributions cannot be isolated without knowing these eigenpairs.

This should clarify the role of the matrix function $h(\mathbf{A})$. It modulates the contributions of different eigenpairs. When preserving only a subset of eigenpairs its role is to cancel the contributions of the other eigenpairs such that $h(\mathbf{A})$ is the linear combination of only the selected eigenpairs. See Section 6.3.2 for details.

Unfortunately, $h(\mathbf{A})$ is a $N \times N$ matrix even when all but e eigenpair contributions are eliminated by the matrix function. In order to achieve a dimensionality reduction random projections are used. Random projections are a technique based on a lemma by Johnson and Lindenstrauss (1984) that enables the embedding of high-dimensional data in low dimensional spaces by projection onto random unit vectors. This method does not require any orthogonalization and preserves pairwise distances with high probability.

Suppose each row of $h(\mathbf{A})$ is embedded in e dimensions and let $\mathbf{F} \in \mathbb{R}^{N\times e}$ be a matrix of random column vectors

$$\mathbf{F} = [\, f_1 \; f_2 \; \cdots \; f_e \,],$$

with each column vector f containing N independent Gaussian random variables distributed as

$$f(i) \sim N\left(0, \frac{1}{e}\right), \qquad \forall i \in \{1, \ldots, N\}. \tag{75}$$

The random projection is expressed with the following matrix product,

$$\mathbf{M} = h(\mathbf{A})\mathbf{F}, \tag{76}$$

where $\mathbf{M} \in \mathbb{R}^{N\times e}$. The rows of $\mathbf{M}$ are e-dimensional, randomly rotated version of the rows $h(\mathbf{A})$ and, critically, $\mathbf{M}$ is distributed as $h(\mathbf{A})$ with high probability. See Section 6.3.3 for more details.

As a consequence, the SCG partitioning can be computed with the columns of $\mathbf{M}$ and, as shown in Section 6.4.1, $\mathbf{M}$ can be approximated efficiently with a truncated Chebyshev polynomial series.

The framework presented here enables flexibility in the choice of the function $h : \mathbb{R} \to \mathbb{R}$ to achieve different effects. The function h is sometimes called a filter and different definitions and application examples can be found in Sandryhaila and Moura (2013) and Shuman et al. (2013).

In conclusion, $\mathbf{M}$ is a distance-preserving, low-dimensional approximation of $h(\mathbf{A})$ and its columns are relative distance-preserving approximation of the columns of $\mathbf{U}$ associated to non-zero eigenpairs.

6.3.2 Eigenspace Truncation with Matrix Functions

Consider the eigendecomposition expressed as a sum over the eigenpairs as formalized by Equation (3). The eigenvector product inside the sum is a matrix $\mathbf{P}_i \in \mathbb{R}^{N \times N}$. This matrix is the eigenprojector onto the eigenspace of λ_i. The eigendecomposition can be equivalently written as

$$\mathbf{A} = \sum_{i=1}^{N} \lambda_i \mathbf{P}_i.$$

This form is an expansion of the adjacency matrix as a series of eigenprojectors with the eigenvalues taking the role of expansion coefficients.

Assume that the matrix $\mathbf{A}$ is a variable of a monomial function, for example, $h(\mathbf{A}) = \mathbf{A}^2$. Writing this explicitly shows that only the eigenvalues are affected,

$$\mathbf{A}^2 = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top = \mathbf{U}\mathbf{\Lambda}^2\mathbf{U}^\top = \sum_{i=1}^{N} \lambda_i^2 \mathbf{P}_i.$$

This behavior generalizes to polynomial functions in matrix variables. One definition of a *matrix function* from Higham (2008, Chapter 1) is the following. Let $h(y)$ be a function of a scalar variable $y \in \mathbb{R}$. A corresponding matrix function $h(\mathbf{A})$ is a matrix of the same dimensions as $\mathbf{A}$ and defined on the spectrum of $\mathbf{A}$,

$$h(\mathbf{A}) \coloneqq \mathbf{U}h(\mathbf{\Lambda})\mathbf{U}^\top = \mathbf{U} \begin{pmatrix} h(\lambda_1) & & & \\ & h(\lambda_2) & & \\ & & \ddots & \\ & & & h(\lambda_n) \end{pmatrix} \mathbf{U}^\top = \sum_{i=1}^{N} h(\lambda_i)\mathbf{P}_i. \tag{77}$$

Thus, the matrix function $h(\mathbf{A})$ reduces to the application of a scalar function $h(y)$ to each eigenvalue of $\mathbf{A}$ and a multiplication of the result with the corresponding eigenprojector. Refer to Higham (2008) for a complete definition of matrix functions and a discussion of their properties.

The spectral embedding framework presented in Section 6.3.1 introduces a matrix function in variable $\mathbf{A}$. Recall that its role is to cancel the contributions of all but the e leading eigenpairs and that e is a parameter. Such functions are called filters. The described function is a high-pass filter as it cancels any contribution of eigenpairs whose eigenvalue is below a cut-off value of λ_e.

In ECSPM, a Heaviside function is used for polynomial filtering. The Heaviside matrix function is defined by the scalar step function $h_{\lambda_e} : [\lambda_N, \lambda_1] \to \{0, 1\}$ such that

$$h_{\lambda_e}(y) \coloneqq \begin{cases} 1 & \text{for } y \geq \lambda_e \\ 0 & \text{for } y < \lambda_e. \end{cases} \tag{78}$$

As shown below, this definition leads to $h_{\lambda_e}(\mathbf{A})$ being equivalent to an eigenprojector onto the eigenspaces associated to the leading e eigenvectors,

$$\begin{aligned} h_{\lambda_e}(\mathbf{A}) &= \mathbf{U} h_{\lambda_e}(\mathbf{\Lambda}) \mathbf{U}^\top \\ &= \mathbf{U} \begin{pmatrix} \mathbf{I}_e & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \mathbf{U}^\top \\ &= \sum_{i=1}^{e} u_i u_i^\top = \mathbf{U}_e \mathbf{U}_e^\top = \mathbf{P}_e. \end{aligned} \tag{79}$$

The equality of the sum expression to $\mathbf{U}_e \mathbf{U}_e^\top$ is given by the equality of Equations (2) and (3). Equivalent filter functions are defined in many related works, for example, Di Napoli et al. (2016), Paratte and Martin (2016), and Tremblay et al. (2016b).

With this definition of the filter function, Equation (74) becomes

$$\widehat{\mathbf{V}}_e = [\ u_1\ \cdots\ u_e\ \mathbf{0}\ \cdots\ \mathbf{0}\ ]. \tag{80}$$

Except for the dimensions, $\widehat{\mathbf{V}}_e \in \mathbb{R}^{N \times N}$ and $\mathbf{U}_e \in \mathbb{R}^{N \times e}$ are equivalent. The rows of $\widehat{\mathbf{V}}$ as defined in Equation (74) preserve the *relative* distances in respect to the rows of $\mathbf{U}$. When using Equation (80), these distances are preserved *exactly* for the first e components.

As explained in Section 6.3.1, this means $h_{\lambda_e}(\mathbf{A})$ also preserves the first e eigenvectors exactly because the projection $\widehat{\mathbf{V}}_e \mathbf{U}^\top$ is a rotation (change of basis) that does not change the distribution of entries.

The random projection defined in Equation (76) becomes

$$\mathbf{M}_e = h_{\lambda_e}(\mathbf{A}) \mathbf{F} = \mathbf{P}_e \mathbf{F}. \tag{81}$$

Note, this is equivalent to the orthogonal projection of the random column vectors onto the subspace spanned by the first e eigenvectors. The eigenprojector in (79) is an orthogonal projector (see Section 2.3.3) that preserves all distances exactly even without considering the probabilistic guarantees of the Johnson-Lindenstrauss lemma. This should not be surprising; by construction $h_{\lambda_e}(\mathbf{A})$ must have rank e because it is the linear combination of only e orthogonal vectors. Clearly, it must be possible to project each of its rows onto an e dimensional subspace.

6.3.3 Distance Preserving Random Projections

Equation (81) suggests, that the random projection is distance preserving by virtue of the matrix function being equal to an orthogonal projector. In Section 6.4, approximation

methods for $h_{\lambda_e}(\mathbf{A})$ are defined. This means that in practice, the matrix function will not represent the distribution of $\mathbf{U}_e$ exactly due to some approximation error. Therefore, $h_{\lambda_e}(\mathbf{A})$ may not actually be orthogonal. Nevertheless, the random projection preserves whatever distribution $h_{\lambda_e}(\mathbf{A})$ has due to the properties of the *Johnson-Lindenstrauss lemma*. This is shown in the following paragraphs.

The matrix function $h_{\lambda_e}(\mathbf{A})$ is always of dimensionality $N \times N$. Therefore, the number of rows equals their dimensionality. However, the following result is more general, thus the projection is explained with m arbitrary vectors $w \in \mathbb{R}^N$. They can be replaced with the rows of $h_{\lambda_e}(\mathbf{A})$; in which case $m = N$.

Theorem 6.1 (Distributional Johnson-Lindenstrauss lemma, adapted). *Given a set W containing m points in $\mathbb{R}^N$ and any $0 < \epsilon < 1$. Let e be a positive integer such that $e \geq 4(\epsilon^2/2 - \epsilon^3/3)^{-1} \log m$ and $\mathbf{F} \in \mathbb{R}^{N \times e}$ with each $F_{ij} \sim N(0, 1/e)$. Then, for every pair $(w_i, w_j) \in W$,*

$$(1 - \epsilon)\|w_i - w_j\|^2 \leq \|w_i\mathbf{F} - w_j\mathbf{F}\|^2 \leq (1 + \epsilon)\|w_i - w_j\|^2$$

with probability at least $1 - O(1/m^2)$.

The parameter ϵ controls the accuracy of the approximation arbitrarily. The matrix $\mathbf{F}$ is defined as in Equation (75) and the variance is set such that the projected rows preserve their length or 2-norm (see below).

Theorem 6.1 has been proven by Johnson and Lindenstrauss (1984) but with a different distribution on $\mathbf{F}$ and using a set of strictly orthogonal random vectors. Subsequently, this lemma has been improved by relaxing the dependence on a strictly orthogonal set of vectors to vectors only orthogonal in expectation. The version presented above is proven formally in Dasgupta and Gupta (1999). However, further extensions and improvements have been proposed, for example, by Achlioptas (2003).

The sketch proof below should illustrate that a random projection corresponds to a random linear combination of the projected vector and that choosing the variance as $1/e$ preserves the norm (or length). The intuition behind the proof is that that projecting an arbitrary fixed vector $w \in \mathbb{R}^N$ onto e random unit vectors means each of the e projected components is a linear combination of w with N random variable picked from $N(0, 1)$. Each random variable has, in expectation, a value of ± 1 (the standard deviation). As a consequence, the projection does not scale the components. Therefore, projecting w from N to e dimensions means the projected vector consist of e random linear combinations and its squared expected norm becomes $(e/N)\|w\|^2$. This length distortion is corrected by picking the random variables from $N(0, 1/e)$. Therefore, the expected length of the projection is that of the projected vector w.

Sketch Proof of Theorem 6.1. Let any $F'_{ij} \sim N(0,1)$. Note that $F_{ij} = (1/\sqrt{e})F'_{ij}$.

$$\begin{aligned} E\left[\|wF\|^2\right] &= E\left[\sum_{j=1}^{e}\left(\sum_{i=1}^{N}\frac{1}{\sqrt{e}}w(i)F'_{ij}\right)^2\right] \\ &= \sum_{j=1}^{e}\frac{1}{e}\,E\left[\left(\sum_{i=1}^{N}w(i)F'_{ij}\right)^2\right] \\ &= \sum_{j=1}^{e}\frac{1}{e}\sum_{i=1}^{N}w(i)^2E[F'^2_{ij}] \end{aligned}$$

From this equality one needs to see that $F'_{ij} \sim N(0,1)$ implies that its squared expected value is equal to the variance, which is 1. Thus the expected length can be written as

$$E\left[\|wF\|^2\right] = \sum_{j=1}^{e}\frac{1}{e}\sum_{i=1}^{N}w(i)^2 = \|w\|^2.$$

□

F is used as a projection matrix onto a random subspace that is not required to be strictly orthonormal. As shown by Indyk and Motwani (1998) as well as Dasgupta and Gupta (1999), choosing the entries independently from a Gaussian distribution with zero mean and variance 1 leads to orthogonality and normality in expectation. The columns of **F** are, by construction, *almost* unit vectors in $\mathbb{R}^N$ pointing in random *almost* orthogonal directions from the origin and spanning an e-dimensional hyperplane.

One may wonder why not to simply choose e random coordinates from each point to obtain a random embedding. Recall that the projection needs to preserve length. When the vectors representing points in W have most of the length contribution concentrated among only few components, these components are likely to be missed by a uniformly random selection. For example, the vectors $e_1 = (1,0,0,\ldots,0)$ and $e_2 = (0,0,\ldots,0,1)$ are both unit length. However, picking few of theirs components at random for an embedding has a high chance of selecting only zeroes and thus severely underestimating their length. Random projections distribute length concentrations over all N dimensions which means that the length contribution that is missed when picking only e out of N components can be predicted. This is what the sketch proof shows. In summary, the projection is a mechanism introducing random rotations to randomly distribute the components of a projected vector such that random omissions have a predictable effect.

An informal interpretation of Theorem 6.1 in respect to the ECSPM framework is the following. All N rows of $h_{\lambda_e}(\mathbf{A})$ can be embedded into a subspace of dimensionality $e \geq O(\log N)$ while preserving their pairwise distances with high probability.

In conclusion, as long as $e \geq O(\log N)$, the random projections are not expected to cause a large distortion. Note that this bound allows for an exponential reduction of e in respect to N. In general, e will be at least an order of magnitude larger than this bound.

6.3.4 Efficient and Controllable SPM (ECSPM)

As explained in the introduction to this section, ECSPM uses the same process as CGSPM. The process is defined in Section 5.3, Figure 5.1. The input to the process is a graph $G(V, E)$ of size N and the following parameters:

k An integer such that $0 < k \leq N$. The size of the coarse-grained graph. Specifically, the rank of the SCG projector $\mathbf{P}$ such that the coarse-graining projection $\mathbf{PAP}$ is an orthogonal projection onto $\mathbb{R}^k$.

e An integer such that $0 < e \leq k$. The number of eigenpairs to preserve in the coarse-graining. Specifically, it controls the number of eigenspaces that are partitioned to create a SCG projector $\mathbf{P}$. An accurate partitioning implies the preservation of the e leading eigenvectors of $\mathbf{A}$ in the projection $\mathbf{PU}_e$ such that $\|\mathbf{U}_e - \mathbf{PU}_e\|$ is minimized according to Equation (63). This parameters should not be larger than k because the coarse-graining projection has $\text{rank}(\mathbf{P}) = k$ (see Section 4.2) and cannot preserve more than k orthogonal dimensions. Informally, as e approaches k, the link-prediction can be expected to be more accurate.

z An integer such that $0 < z \leq N$. The number of random vectors used to approximate λ_e. More random vectors increase the probability of an accurate approximation. An inaccurate approximation of λ_e implies an inaccurate coarse-graining projector $\mathbf{P}$. Choosing $z = e$ leads to a high probability of an accurate approximation. On the other hand, as discussed in Section 6.4.3, inaccuracies in this approximation may not be very detrimental.

b An integer such that $b > 0$. The degree for the Chebyshev polynomial expansion used to approximate the matrix function $h_{\lambda_e}(\mathbf{A})$ (see Equation 87). A larger value approximates the function more precisely (see Section 6.4.3).

d An integer such that $d > 0$. The degree of the Chebyshev polynomial approximation of the filter function used for the approximation of λ_e. It has the same effect as parameter b but allows to approximate the eigenvalue λ_e with different polynomial degree.

The input graph is fully represented by its adjacency matrix is $\mathbf{A} \in \mathbb{R}^{N \times N}$. ECSPM only changes how the eigenspace partitioning is computed. The changed steps are described below. Their implementation is defined in Section 6.4.

The graph vertices of the input graph are partitioned into k groups which yields a partitioning Γ such that $|\Gamma| = k$. The computation consists of the following steps.

1. The largest eigenvalue λ_1 and the smallest eigenvalue λ_N of $\mathbf{A}$ are estimated with a standard iterative eigensolver and a small number of iterations. Only low accuracy is required. Therefore, this is very efficient.
2. The eigenvalue λ_e of $\mathbf{A}$ is approximated with a truncated Chebyshev polynomial expansion filtering as proposed by Tremblay et al. (2016b) and Paratte and Martin (2016). This process is defined in Section 6.4.2 and Algorithm 6.2.
3. The matrix $\mathbf{M} \in \mathbb{R}^{N \times e}$ is computed with truncated Chebyshev polynomial expansion filtering as defined in Section 6.4.1 and Algorithm 6.3.
4. K-means is used to partition all the N vertices of G into k groups as defined in Section 4.3.4. The clustering uses each row of $\mathbf{M}$ as a feature vector for the corresponding vertex. The result is a partitioning Γ.

5. The SCG semi-projectors $\mathbf{R}$ and $\mathbf{L}$ are computed from Γ with Equations (60) and (61) (see Section 4.2).

All Algorithms that formalize this process are defined in Section 6.4.4.

The remaining link prediction process proceeds as follows. The perturbation set $\mathbf{\Delta A}$ is selected from the input matrix and the unperturbed matrix $\mathbf{A^r} = \mathbf{A} - \mathbf{\Delta A}$ is obtained. Then, the unperturbed matrix $\tilde{\mathbf{A}}^{\mathbf{r}} = \mathbf{L}\mathbf{A}^{\mathbf{r}}\mathbf{R}^\top$ as well as the perturbation set $\mathbf{\Delta}\tilde{\mathbf{A}} = \mathbf{L}(\mathbf{\Delta A})\mathbf{R}^\top$ are coarse-grained and used to compute a link-existence estimator $\hat{\mathbf{A}}$ with SPM. Refer to Section 5.3 for a precise description of the link prediction steps.

6.4 Computational Aspects

This section discusses matters related to the computation of the eigenspace partitioning in ECSPM. In particular, the efficient approximation of the random projection matrix $\mathbf{M}$ as well as that of the eigenvalue λ_e with a truncated Chebyshev polynomial expansion is defined. It includes a discussion of the approximation error to provide intuition about the behavior of the approximation and the effect of method parameters. Afterwards, the computational cost and complexity of the ECSPM method is analyzed.

6.4.1 Approximation of the Random Projection Matrix

The key to computational efficiency is the approximation of matrix $\mathbf{M}$ with polynomial expansion filtering. In this section, this approximation is defined with Chebyshev polynomials and the filter $h_{\lambda_e}(\mathbf{A})$ defined by Equations (78) and (79). Chebyshev polynomials and Jackson smoothing for ordinary functions are defined and explained in Section 2.6.

In a first step, the matrix function $h_{\lambda_e}(\mathbf{A})$ is expressed as a Chebyshev polynomial series. Then, the projection $\mathbf{M} = h_{\lambda_e}(\mathbf{A})\mathbf{F} = [\, h_{\lambda_e}(\mathbf{A})f_1 \;\cdots\; h_{\lambda_e}(\mathbf{A})f_e \,]$, where f_i denotes the i-th column of $\mathbf{F}$, is expressed with Chebyshev polynomials.

The Chebyshev-Jackson polynomial expansion of matrix function $h_{\lambda_e}(\mathbf{A})$ is defined as

$$h_{\lambda_e}(\mathbf{A}) \coloneqq \sum_{i=0}^{\infty} g_i c_i \overline{T}_i(\mathbf{A}). \tag{82}$$

Each matrix polynomial $\overline{T}_i(\mathbf{A}) \in \mathbb{R}^{N\times N}$ is defined as

$$\overline{T}_i(\mathbf{A}) \coloneqq \mathbf{U} \begin{pmatrix} \overline{T}_i(\lambda_1) & & \\ & \ddots & \\ & & \overline{T}_i(\lambda_N) \end{pmatrix} \mathbf{U}^\top,$$

where the terms $\overline{T}_i(\lambda_j)$ denote the shifted Chebyshev polynomial of the first kind of degree i. These terms are defined by Equation (32).

Let the shifted eigenvalues be $\bar{\lambda}_i = (\lambda_i - \alpha_1)/\alpha_2$ with the constants α_1 and α_2 defined by Equation (31). According to Schofield et al. (2012), the Chebyshev coefficients c_i for a

high-pass filter on the interval $[\lambda_e, \lambda_1]$ are defined by

$$c_i = \begin{cases} \frac{1}{\pi}\arccos(\bar{\lambda}_e) - \arccos(\bar{\lambda}_1) & \text{for } i = 0 \\ \\ \frac{2}{\pi i}\left[\sin(i\arccos(\bar{\lambda}_e)) - \sin(i\arccos(\bar{\lambda}_1))\right] & \text{for } i > 0. \end{cases} \tag{83}$$

The Jackson smoothing coefficients g_i are defined in Section 2.6.4.

Equation (82) is never fully materialized as this matrix can be very large and dense. It is a well known optimization to avoid this type of interim matrix product by computing $h_{\lambda_e}(\mathbf{A})f_i$ directly (e.g., Greenbaum and Trefethen, 1994, Section 2). This product can be computed independently for each $i \in \{1, \ldots, e\}$. The following derivation expands upon the definition in Hammond et al. (2011) to show in detail how the recurrence relation of Chebyshev polynomials can be exploited to avoid the computation of $\mathbf{U}$ or $h_{\lambda_e}(\mathbf{A})$ entirely.

Consider the product $\overline{T}_i(\mathbf{A})f$ with f representing any column of $\mathbf{F}$,

$$\overline{T}_i(\mathbf{A})f = \mathbf{U}\begin{pmatrix} \overline{T}_i(\lambda_1) & & \\ & \ddots & \\ & & \overline{T}_i(\lambda_N) \end{pmatrix}\mathbf{U}^\top f.$$

Recall that the first two Chebyshev polynomials are defined $T_0(x) = 1$ and $T_1(x) = x$ (see Section 2.6.1). Applying Equation (32) to derive the first shifted polynomials results in $\overline{T}_0(y) = 1$ and $\overline{T}_1(y) = (x - \alpha_1)/\alpha_2$. As a consequence, the matrix-vector product for the shifted matrix polynomial of degree 0 is

$$\overline{T}_0(\mathbf{A})f = \mathbf{U}\mathbf{U}^\top f = f \tag{84}$$

and the product for the shiftet matrix polynomial of degree 1 becomes

$$\begin{aligned} \overline{T}_1(\mathbf{A})f &= \mathbf{U}\begin{pmatrix} \frac{\lambda_1-\alpha_1}{\alpha_2} & & \\ & \ddots & \\ & & \frac{\lambda_N-\alpha_1}{\alpha_2} \end{pmatrix}\mathbf{U}^\top f \\ &= \frac{1}{\alpha_2}\mathbf{U}\left[\begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_N \end{pmatrix} - \alpha_1\mathbf{I}_N\right]\mathbf{U}^\top f \\ &= \frac{1}{\alpha_2}\left[\mathbf{U}\begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_N \end{pmatrix}\mathbf{U}^\top - \mathbf{U}\mathbf{U}^\top\alpha_1\mathbf{I}_N\right]f \\ &= \frac{1}{\alpha_2}(\mathbf{A} - \alpha_1\mathbf{I})f. \end{aligned} \tag{85}$$

The first equation and the third equality in the second equation make use of the orthogonality of $\mathbf{U}$, that is, $\mathbf{U}\mathbf{U}^\top = \mathbf{I}_N$.

All remaining polynomials of degree $i \geq 2$ can be obtained from the recurrence relation defined by Equation (25). Using (85) allows to write the remaining shifted Chebyshev matrix polynomials as

$$\overline{T}_i(\mathbf{A})f = \frac{2}{\alpha_2}(\mathbf{A} - \alpha_1\mathbf{I})\overline{T}_{i-1}(\mathbf{A})f - \overline{T}_{i-2}(\mathbf{A})f \quad \text{for } i \geq 2. \tag{86}$$

To summarize, the products $\overline{T}_0(\mathbf{A})f$ and $\overline{T}_1(\mathbf{A})f$ are defined immediately given only $\mathbf{F}$ and $\mathbf{A}$. All remaining products can be computed with (86). Therefore, the recurrence relation makes it possible to avoid the diagonalization of $\mathbf{A}$. All columns of $\mathbf{M}$ are defined using only the matrix vector products. An order b approximation of $\mathbf{M}$ can be computed by truncating (82) after degree b,

$$\mathbf{M} \approx \sum_{i=0}^{b} g_i c_i \overline{T}_i(\mathbf{A})\mathbf{F}. \tag{87}$$

6.4.2 Eigenvalue Estimation

The approximation of $\mathbf{M}$ can be computed without knowledge of any eigenvectors. Nevertheless, three eigenvalues are still required. The extreme eigenvalues λ_1 and λ_N determine the constants α_1 and α_2 which are defined by Equation (31) and used to shift the domain of the Chebyshev polynomials into the range $[\lambda_e, \lambda_1]$. Conservative approximations of these eigenvalues are sufficient because no precise normalization is required as long as the estimates include the full interval. Therefore, these eigenvalues can be be efficiently approximated by a few iterations of the Lanczos method. Refer to Section 6.4.4 for a detailed specification.

On the other hand, a more accurate approximation of eigenvalue λ_e is required when $\mathbf{M}$ should reflect the distribution of $\mathbf{U}_e$. An efficient method to estimate such interior eigenvalues has been proposed in Paratte and Martin (2016) based on an eigenvalue count method by Di Napoli et al. (2016). The latter method estimates the number of eigenvalues on an arbitrary interval with polynomial expansion filtering and Chebyshev polynomials. Therefore, an estimation of the number of eigenvalues in an interval can be computed with the same approach as the approximation of $h_{\lambda_e}(\mathbf{A})$ described in Section 6.4.1.

Equation (79) established that the matrix function $h_{\lambda_e}(\mathbf{A})$ is equal to the eigenprojector $\mathbf{P}_e$. By definition, all eigenvalues of projection matrices are either 0 or 1 and by construction the non-zero eigenvalues of $\mathbf{P}_e$ must correspond to the e leading eigenpairs. Furthermore, the following definition of the trace operation is used,

$$\text{trace}(\mathbf{P}_e) = \text{trace}(h_{\lambda_e}(\mathbf{A})) = \sum_{i=1}^{N} \lambda_i = e.$$

Consider a naive process to estimate λ_e. Suppose λ_1 and λ_N have been estimated conservatively. One can pick an estimated λ'_e at some value in the interval $[\lambda_N, \lambda_1]$ and approximate the corresponding matrix function $h_{\lambda'_e}(\mathbf{A})$ as described in Section 6.4.1.

Because the filter function is defined to preserve only the eigenvalues from λ'_e to λ_1, one can compare $\text{trace}(h_{\lambda'_e}(\mathbf{A})) = e'$ to the searched value e. If the $e' > e$, then interval $[\lambda'_e, \lambda_1]$ is too large and the estimate λ'_e has to be chosen closer to λ_1. On the other hand, if the trace is too small, the estimate has to be chosen closer to λ_N in order to enlarge the interval. In this manner the value for λ'_e can be adjusted until $\text{trace}(h_{\lambda'_e}(\mathbf{A})) = e$ which implies that $\lambda'_e = \lambda_e$.

A first optimization to this naive process is to avoid the materialization of $h_{\lambda'_e}(\mathbf{A})$. Matrix traces can be approximated with Hutchinson's stochastic estimator. Very similar to random projections, it exploits the statistical properties of random vectors with zero-mean for the approximation of the quadratic form $f^\top \mathbf{P}_e f$. In expectation, this product converges to the trace of $\mathbf{P}_e$ (Hutchinson, 1990). In the original formulation, the components of vector f are Rademacher random variables. Di Napoli et al. (2016) prefer to use a variant with Gaussian random variables such that each vector component $f(j) \sim N(0, 1)$ for $j = 1, \ldots, N$ and each f is normalized to unit length. In this case, the trace of $\mathbf{P}_e$ is approximated by the expected value over z sample vectors

$$\text{trace}(\mathbf{P}_e) \approx \frac{N}{z} \sum_{i=1}^{z} f_i^\top \mathbf{P}_e f_i.$$

Substituting $\mathbf{P}_e$ with the Chebyshev-Jackson polynomial series given by Equation (82) and truncating the series after degree d approximates the trace of the eigenprojector using only matrix-vector products,

$$\text{trace}(\mathbf{P}_e) \approx \frac{N}{z} \sum_{i=1}^{z} \sum_{j=0}^{d} f_i^\top g_j c_j \overline{T}_j(\mathbf{A}) f_i = \frac{N}{z} \sum_{i=1}^{z} f_i^\top \mathbf{M}^d(i), \tag{88}$$

where $\mathbf{M}^d$ is defined by (87) but truncated at polynomial order d instead of b.

A second improvement over the naive process is to use a more advanced search algorithm. In Tremblay et al. (2016b), a dichotomic search is proposed that required $O(\log N)$ iterations. Paratte and Martin (2016, Algorithm 2) propose an improved algorithm for the Laplacian spectrum that requires only a constant number of iterations. Algorithm 6.2 formalizes the latter algorithm as implemented in ECSPM. It is slightly modified to adapt it to the adjacency matrix spectrum which has reversed order and negative eigenvalues.

6.4.3 Approximation Error

Two different types of approximations can introduce an approximation error in the computation of $\mathbf{M}$. First, the approximation of the matrix function $h_{\lambda_e}(\mathbf{A})$ has an accuracy that depends on the order of the expansion. This is governed by the parameter b for Equation (87) and d for Equation (88). Furthermore, the accuracy depends on the quality of the approximated λ_e value. The second approximation error is introduced by the random projection and the stochastic estimator of the matrix trace. In the following paragraphs, approximation errors are discussed to provide an intuition about how they can be influenced with the method parameters.

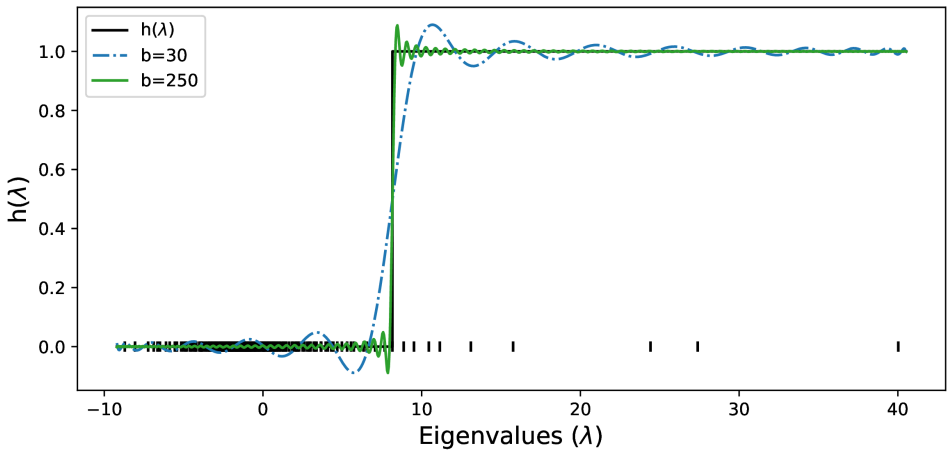


Fig. 6.1: *The x-axis represents the spectrum of the* jazz *graph. The ideal filter $h(y)$ returns 1 when $y \geq \lambda_{10}$ and 0 otherwise. The standard Chebyshev polynomial approximation of different degrees exhibit the Gibbs phenomenon around the discontinuity at λ_{10}.*

The first type of approximation error can be influenced by the degree of the polynomial approximation, that is, the parameter b or d. Ideally, the approximation returns exactly 1 for all eigenvalues in the interval $[\lambda_e, \lambda_1]$ and 0 otherwise. Figure 6.1 depicts the ideal filter (black step function) and two truncated Chebyshev polynomial series with different degree. The large oscillations at the discontinuity are called Gibbs phenomenon. The oscillation causes an error that does not disappear with increasing polynomial order (see Section 2.6.4). For the approximation error, the general shape of the oscillations does not matter but the values the function takes at the location of each eigenvalue (ticks on the x-axis) matters. Depending on the distribution of the eigenvalues, some have a larger approximation error than others. The main issue is that these oscillations can disturb the relative importance of some eigenpairs substantially and cause errors in the approximation of $h_{\lambda_e}(\mathbf{A})$.

The Gibbs phenomenon is corrected with Jackson smoothing as shown in Figure 6.2. With these corrections, the error is better behaved and does not disturb the relative importance of the eigenpairs. The number of affected eigenpairs depends on the order of the expansion and the distribution of the eigenvalues. When the discontinuity is located in a tightly clustered region, many eigenpairs whose contributions should be filtered out will continue to contribute to the result. In ECSPM, this is not necessarily a big problem because the filtering is done primarily to save computational cost. Some contributions from other eigenspaces may not be that harmful to the eigenspace partitioning. Furthermore, $h_{\lambda_e}(\mathbf{A})$ is not used for link prediction but only for SCG. Therefore, these approximation errors do not directly affect the link prediction process.

A similar situation occurs when the estimation of λ_e is wrong. Then, the approximated discontinuity is shifted and some eigenpairs end up on the wrong side of the step. This is more likely to happen when λ_e is located in a densely clustered region. Again, because $h_{\lambda_e}(\mathbf{A})$ is used only for eigenspace partitioning this may not be a big problem as long as the

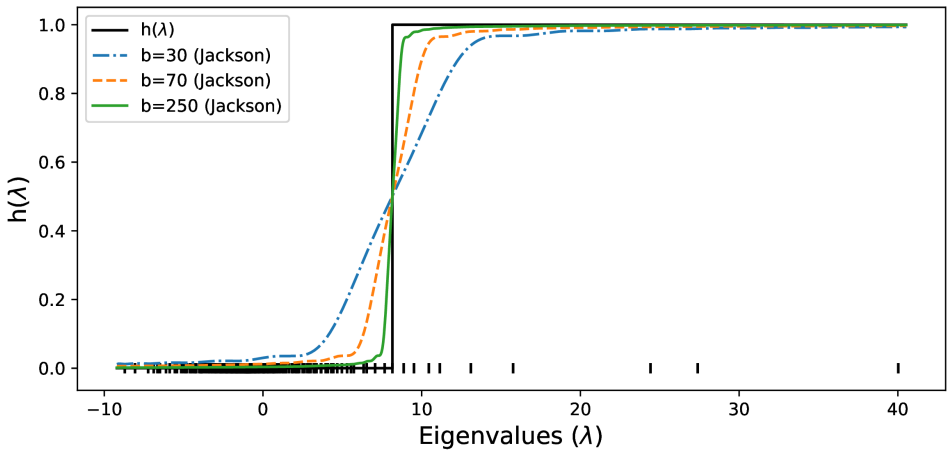


Fig. 6.2: *Jackson smoothing removes the Gibbs oscillations. The Chebyshev-Jackson approximation of higher degree approximate the ideal step function better.*

estimation of λ_e is not completely wrong. Furthermore, the quality of the approximation of λ_e depends primarily on d, the order of the polynomial expansion used for its estimation. Thus, choosing an appropriate order of approximation should prevent a large error.

The second type of approximation error concerns the errors caused by projections onto random vectors. When the approximation of $h_{\lambda_e}(\mathbf{A})$ is very accurate, it is an almost orthogonal projector and the random projection error can be neglected. On the other hand, if $h_{\lambda_e}(\mathbf{A})$ is perturbed significantly, the probability guarantees of Theorem 6.1 (Johnson-Lindenstrauss lemma) state that the distribution of $h_{\lambda_e}(\mathbf{A})$ is preserved in $\mathbf{M}$ with probability $1 - O(1/N^2)$ as long as $e \geq O(\log N)$. This is negligible in practice when e is chosen large enough.

A similar bound is given in Di Napoli et al. (2016, Lemma 2.1) for Hutchinson's trace estimator and the number of random vectors z used for its approximation. A loose bound to reach convergence is $z = O(e)$. However, in practice the value can often be smaller and, as discussed above, a small error in the estimation of λ_e may not actually be that harmful to ECSPM.

6.4.4 Algorithms

This section lists and describes the algorithms implemented for ECSPM. These algorithms only change the implementation of the GETPROJECTORS function call on Line 4 in Algorithm 5.2. All other algorithmic steps are defined in Section 5.4.1. The notation and functions used in all algorithms are defined in Appendix A.

ECSPM Projectors: The ECSPM projectors are computed as defined by Algorithm 6.1.

On Line 1, a partial eigendecomposition is computed to approximate the largest eigenvalue λ_1 and the smallest eigenvalue λ_N of $\mathbf{A}$. The estimates $\lambda_{\min}$ and $\lambda_{\max}$ do not need to be precise and can be computed efficiently with the IRLM algorithm by setting the relative precision for convergence to a large value. This leads to very fast convergence

within few iterations of the Lanczos method (see Section 2.4). A tolerance of 0.5% relative to machine precision is used in ECSPM. This value is adopted from the GSPBOX implementation of the Chebyshev polynomial approximation (Perraudin et al., 2014).

On Lines 2 and 3, the extremal eigenvalue estimates are adjusted to ensure that they contain the interval $[\lambda_N, \lambda_1]$ fully. 1% is added to $\lambda_{\max}$ and 1% is subtracted from $\lambda_{\min}$. This ensures that $\lambda_{\max} \geq \lambda_1$ and $\lambda_{\min} \leq \lambda_N$. Again, these values were adopted from Perraudin et al. (2014).

On Line 7, $\mathbf{M}$ is partitioned into k groups with k-means and k-means++ initialization. This partitioning method is described in Section 4.3.4.

Afterwards, the SCG semi-projectors are computed from the partitioning Γ as defined in Section 4.2.1.

Algorithm 6.1 ECSPM Projectors

Input: $\mathbf{A}, N, e, b, d, z, k,$ maxIter

1: $\lambda_{\min}, \lambda_{\max} \leftarrow \text{EIGSHVAL}(A, [1, N])$ ▷ *Tolerance: 0.5%*
2: $\lambda_{\min} \leftarrow \lambda_{\min} - |\lambda_{\min}| * 0.01$ ▷ *Subtract 1%*
3: $\lambda_{\max} \leftarrow \lambda_{\max} + |\lambda_{\max}| * 0.01$ ▷ *Add 1%*
4: $\lambda_e \leftarrow \text{ESTIMATEEIGENVALUE}(\mathbf{A}, \lambda_{\min}, \lambda_{\max}, e, N, d, z, \mathit{maxIter})$ ▷ *Alg. 6.2*
5: $\mathbf{F} \leftarrow \text{RANDN}(N, e)/\sqrt{e}$
6: $\mathbf{M} \leftarrow \text{FILTER}(\mathbf{A}, \mathbf{F}, b, \lambda_e, \lambda_{\max}, \lambda_{\min}, \lambda_{\max})$ ▷ *Alg. 6.3*
7: $\Gamma \leftarrow \text{K-MEANS}(\mathbf{M}, k)$
8: $\mathbf{R} \leftarrow \text{SCGPROJECTOR}(\Gamma)$ ▷ *Eq.* (60)
9: $\mathbf{L} \leftarrow \mathbf{R}$ ▷ *Eq.* (61)
10: ***return*** $\mathbf{L}, \mathbf{R}$

Estimation of λ_e: Algorithm 6.2 is a modification of the eigenvalue estimation algorithm proposed in Paratte and Martin (2016, Algorithm 2). The original algorithm estimates the k-th smallest eigenvalue in the interval $[0, \lambda_k]$ which corresponds to the k smallest eigenvalues of the Laplacian graph matrix. The modified algorithm searches for the e-th largest eigenvalue in an arbitrary interval.

The algorithm is very similar to a binary search but instead of halving each interval it assumes the eigenvalues are uniformly distributed between the lower and upper bound of the search interval. As a consequence, the location of λ_e is initially estimated at $e * (\lambda_{\max} - \lambda_{\min})/N$ (Line 6). After the number of eigenvalues in this interval is estimated by computing the trace of the eigenprojector (Line 8), the upper bound and the lower bound of the search interval are adjusted and all eigenvalues that have not yet been found are estimated with the same assumption of uniform distribution (Line 22). Only if the search does not progress in one iteration, a binary search step is made (Line 16). The algorithm converges when the estimated eigenvalue count is equal to e or when the number iterations equals the *maxIter* parameter.

This algorithm works reliably even though eigenvalues are generally not distributed uniformly. Refer to Paratte and Martin (2016, Section 4.1) for a detailed discussion.

On Line 8, Equation (88) is implemented. For convenience of notation, the trace approximation is written as $(N/z) * \text{trace}(\mathbf{F}^\top \mathbf{M}^d)$ and then rounded to the nearest integer.

However, the trace operation requires only the z diagonal entries of the $z \times z$ matrix $\mathbf{F}^\top \mathbf{M}^d$ even though the notation implies that the matrix is fully computed. The actual implementation avoids the unnecessary computation of the off-diagonal elements and therefore complies with Equation (88).

Algorithm 6.2 Estimate λ_e (Adapted from Paratte and Martin (2016, Algorithm 2))

Input: $\mathbf{A}, \lambda_{\min}, \lambda_{\max}, e, N, d, z, \text{maxIter}$

$\mathbf{F} \leftarrow \text{RANDN}(N, z)$ ▷ *Random matrix with elements from* $N(0,1)$
for $i \leftarrow 1; i <= z; i \leftarrow i+1$ ***do***
  $\mathbf{F}(i) \leftarrow \mathbf{F}(i)/\|\mathbf{F}(i)\|$ ▷ *Normalize each column to unit length.*
$c_l \leftarrow 0,\ c_u \leftarrow N$
$\lambda_l \leftarrow \lambda_{\min},\ \lambda_u \leftarrow \lambda_{\max}$
$\lambda_e \leftarrow \lambda_u - e\frac{\lambda_u - \lambda_l}{N}$
for q ***from*** 0 ***to*** *maxIter* ***do***
  $c_{est} \leftarrow \lfloor \frac{N}{z} \text{trace}(\mathbf{F}^\top \text{FILTER}(\mathbf{A}, \mathbf{F}, d, \lambda_e, \lambda_{\max}, \lambda_{\min}, \lambda_{\max})) \rceil$ ▷ *Eq.* (88)
  if $c_{est} < e$ ***then***
    $\lambda_l \leftarrow \lambda_e$
  else if $c_{est} > e$ ***then***
    $\lambda_u \leftarrow \lambda_e$
  else
    break
  if $c_l = c_{est}$ *or* $c_u = c_{est}$ ***then***
    $\lambda_e \leftarrow \lambda_u - \frac{\lambda_u - \lambda_l}{2}$
  else
    if $c_{est} < e$ ***then***
      $c_l \leftarrow c_{est}$
    else
      $c_u \leftarrow c_{est}$
    $\lambda_e \leftarrow \lambda_u - (e - c_l)\frac{\lambda_u - \lambda_l}{c_u - c_l}$
return λ_e

Polynomial Expansion Filtering: Algorithm 6.3 has been adapted from the function CHEBY_OP in Perraudin et al. (2014). It computes the matrix $\mathbf{M}$ directly as defined by Equation (87).

Parallelization and Distribution of the Algorithms: In the scope of this work, Algorithms 6.1, 6.2 and 6.3 are sequential implementations not optimized for distributed or parallel execution. However, polynomial expansion filtering is employed to speed-up numerical computations (e.g., Greenbaum and Trefethen, 1994, Ramasamy and Madhow, 2015, Shuman et al., 2011) and can be parallelized and distributed.

- Algorithm 6.2 sequentially searches for the value of λ_e on the interval $[\lambda_N, \lambda_1]$. Using standard approaches from other search algorithms, this algorithm can be parallelized. When $\mathbf{A}$ is large, the memory requirements for such a parallel search can be prohibitively large as well because the Chebyshev polynomial approximation cannot naively share memory. When k segments are searched in parallel, space for k copies of $\mathbf{M}$ is required.

Algorithm 6.3 Compute $\mathbf{M} = h_{\lambda_e}(\mathbf{A})\mathbf{F}$ (Adapted from Perraudin et al. (2014))

Input: $\mathbf{A}, \mathbf{F} b, \lambda_l, \lambda_u, \lambda_{\min}, \lambda_{\max}$

$\alpha_1 \leftarrow (\lambda_{\max} + \lambda_{\min})/2$ ▷ *constants for shifted Chebyshev polynomials*
$\alpha_2 \leftarrow (\lambda_{\max} - \lambda_{\min})/2$
$\lambda_l \leftarrow (\lambda_l - \alpha_1)/\alpha_2$
$\lambda_u \leftarrow (\lambda_u - \alpha_1)/\alpha_2$
$c \leftarrow \{c_0, c_1, \ldots, c_d\}^\top$
$\beta \leftarrow \pi/(b+2)$
$g \leftarrow \{g_0, g_1, \ldots, g_d\}^\top$
$c_0 \leftarrow \frac{1}{\pi}\left[\arccos(\lambda_l) - \arccos(\lambda_u)\right]$
$g_0 \leftarrow \frac{\sin(1)\beta}{(b+2)\sin\beta} + \left(1 - \frac{1}{b+2}\right)$
for $i \leftarrow 1; i <= b; i \leftarrow i+1$ ***do***
    $c_i \leftarrow \frac{2}{\pi i}\left[\sin(i\arccos(\lambda_l)) - \sin(i\arccos(\lambda_u))\right]$ ▷ *Chebyshev coefficients, Eq.* (83)
    $g_i \leftarrow \frac{\sin(i+1)\beta}{(b+2)\sin\beta} + \left(1 - \frac{i+1}{b+2}\right)\cos(i\beta)$ ▷ *Jackson coefficients, Eq.* (34)
$\mathbf{TF}_0 \leftarrow \mathbf{F}$ ▷ *Eq.* (84)
$\mathbf{TF}_1 \leftarrow \frac{1}{\alpha_2} * (\mathbf{A} - \alpha_1 \mathbf{I}_N)\mathbf{F}$ ▷ *Eq.* (85)
$\mathbf{M} \leftarrow g_0 * c_0 * \mathbf{TF}_1 + g_1 * c_1 * \mathbf{TF}_0$
$\mathbf{X} \leftarrow \frac{2}{\alpha_2} * (\mathbf{A} - \alpha_1 \mathbf{I}_N)$
for $i \leftarrow 2; i < b; i \leftarrow i+1$ ***do***
    $\mathbf{TF}_i \leftarrow \mathbf{X}\,\mathbf{TF}_{i-1} - \mathbf{TF}_{i-2}$ ▷ *Eq.* (86)
    $\mathbf{M} \leftarrow \mathbf{M} + g_i * c_i * \mathbf{TF}_i$ ▷ *Eq.* (87)
    $\mathbf{TF}_{i-2} \leftarrow \mathbf{TF}_{i-1}$
    $\mathbf{TF}_{i-1} \leftarrow \mathbf{TF}_i$
return $\mathbf{M}$

- Algorithm 6.3 is called in Algorithm 6.2 as well. It can be trivially parallelized by computing each column of $\mathbf{M}$ in parallel without additional memory requirements. This is used by Shuman et al. (2011) to distribute the computation. Ramasamy and Madhow (2015) used the same circumstance to implement a parallel algorithm.

6.4.5 Operation Count

Below, the operation count for the ECSPM algorithms is estimated in terms of floating point operations (flops). Only the algorithms implemented for ECSPM are analyzed. The computational cost of the remaining link prediction process remains unmodified from what is described in Section 5.4.2. Assumptions about the operation count of matrix and vector operations are defined in Section 3.4.2. Furthermore, the execution of a trigonometric functions is assumed to cost only one flop because they can be executed as a single instruction. However, these operations are substantially more expensive in terms of execution time than additions or multiplications.

The sparsity of matrix $\mathbf{A}$ is critical for the performance of ECSPM. This study assumes that sparse matrix operations have a cost proportional to the number of non-zero matrix elements (nnz). A dense matrix has $nnz = N^2$ while a sparse matrix has $nnz = cN$ for a constant $0 < c \ll N$ which corresponds to $nnz = O(N)$. In graph terms, cN is the number of edges (counting both directions).

Computation Steps: The relevant computational steps of the ECSPM algorithms are listed below. Computationally insignificant steps of sub-linear complexity are omitted as they are not relevant for the following discussion.

1. (Alg. 6.1, Line 1) A partial spectral decomposition of $\mathbf{A}$ is computed to obtain two eigenvalues of matrix $\mathbf{A}$ with IRLM (Calvetti et al., 1994). The cost is bounded by $O(N)$ flop for sparse matrices (see Section 2.4). Since the convergence tolerance is very high this is likely a gross overestimate.
2. (Alg. 6.2, Line 1) The generation of random vectors costs zN flop.
3. (Alg. 6.2, Line 3) The normalization of the random vectors costs $2zN$ flop.
4. (Alg. 6.2, Line 8) Only the diagonal elements of the matrix $\mathbf{F}^\top\mathbf{M}^d$ are computed for the trace operation. Refer to Equation (88). The cost of a projection of the Chebyshev approximation onto a single random vector ($e = 1$) implemented as in Step 7; assume $b = d$. Then, the cost is $(2d{+}4)cN{+}(4d{+}10)N$ flop. Another vector product is required to compute the quadratic form which costs $2N$ flop. To compute the trace of $\mathbf{F}^\top\mathbf{M}^d$, the previous steps are repeated z times (number of diagonal entries). Furthermore, a sum over z elements is required. Additionally, the expansion coefficients are computed once as in Step 6 with $b = d$ for $25d$ flop. Assuming the eigenvalue estimation requires m iterations in total, the final cost for this step is $m{*}(z{*}[(2d{+}4)cN{+}(4d{+}12)N]{+}z{+}25d)$ flop.
5. (Alg. 6.1, Line 5) The generation of the random projection vectors costs $2eN$ flop.
6. (Alg. 6.3, Line 10) The computation of the Chebyshev and Jackson coefficients costs approximately $25b$ flop.
7. (Alg. 6.3, Line 13 to 21) The filtering consists of $2ecN + 2N + eN$ flop for $\mathbf{TF}_1$, $2cN + 2N$ flop for $\mathbf{X}$, $2ebcN + ebN$ flop for $\mathbf{TF}_i$, and $3ebN + 5eN$ flop for $\mathbf{M}$. In total, $(2eb + 2e + 2)cN + (4eb + 6e + 4)N$ flop.
8. (Alg. 6.1, Line 7) K-means requires $O(keNi)$ distance calculations to partition N vectors of length e into k partitions in i iterations (Hartigan and Wong, 1979). In general, i is unpredictable. In practice it can be assumed constant and small. Therefore, a total cost of $O(keN)$ flop is assumed for this step.
9. (Alg. 6.1, Line 8) The computation of the semi-projector costs k flops.

To reduce the complexity of the notation, assume $d = b$ and $z = e$. The total cost of the SCG projector computation with ECSPM is

$$\begin{aligned}&(e * [m * (2b + 4) + 2b + 2] + 2)cN\\ +&(e * [m * (4b + 12) + 4b + k + 11] + 5)\ N\\ &+em + (25m + 25)b + k\end{aligned}$$

floating point operations. Ignoring lower order terms and assuming a constant and small m the cost becomes

$$(4eb + 6e + 2)cN + O(ebN) \tag{89}$$

floating point operations.

6.4.6 Complexity Analysis

The link prediction scenario in this thesis assumes sparse input graphs and ECSPM is designed accordingly. For the sake of argument, consider a dense input matrix $\mathbf{A}$ and the order of polynomial approximation, that is, parameter b, constant. When $\mathbf{A}$ has $cN = N^2$, the cost estimate in Equation (89) becomes bounded by $O(eN^2)$. This is the same complexity as that of computing a partial eigendecomposition with state-of-the art solvers (see Section 2.4). There is no efficiency advantage when using ECSPM on dense matrices and the standard solvers are precise up to machine precision while ECSPM introduces considerable approximation error.

In contrast, the partitioning of a sparse matrix is significantly more efficient. A sparse matrix has $cN = O(N)$ and the computational cost of computing an SCG projector is bounded by $O(ebN)$; taking b constant as above, this yields $O(eN)$. Clearly, ECSPM relies on the efficiency of sparse matrix multiplications to gain its efficiency advantage.

The coarse-grained graph size is determined by the parameter k. The parameter e influences the link prediction accuracy by making the coarse-graining projection preserve more eigenpairs. The polynomial approximation error is controlled by the parameter b. The properties of the parameters are discussed in Section 6.3.4 and the polynomial approximation error in Section 6.4.3.

Assuming e and b constant and relatively small shows that the eigenspace partitioning of ECSPM is much more efficient than that of CGSPM. As motivated in the introduction to this chapter, ECSPM utilizes this efficiency gain to increase the parameter e in order to improve prediction accuracy. When $e = k$, the coarse-graining maximally preserves $\mathbf{U}_k$. Because k is also the size of the coarse-graining, it cannot preserve any more eigenpairs because it lacks orthogonal dimensions to represent them. In this regime, the complexity becomes $O(bkN)$.

Intuitively, a large value for b appears desirable because it leads to a precise approximation of the eigenspaces but it cannot become too large as the computational cost depends linearly on this parameter. When $b = k$, the complexity becomes $O(k^2N)$ which the same computing a sparse partial eigendecomposition of $\mathbf{A}$. Therefore, one should choose large values for b only when e is kept small for compensation.

The overall complexity of the link-prediction process with ECSPM remains the same as with CGSPM. The discussion above is concerned only with the eigenspace partitioning. The coarse-grained link-prediction process continues to have a complexity of $O(k^2N)$, primarily due to the coarse-graining projections that remain unmodified in ECSPM. However, k and N can be chosen independently in ECSPM. In CGSPM, their relationship could not be clearly determined which means that the complexity of ECSPM is always as good as the best case of CGSPM. Therefore, the average case complexity of CGSPM may in fact be slightly higher.

6.5 Experiments

The experiments described in this section use the same experimental setup and methodology as SPM (Lü et al., 2015) and CGSPM (see Section 5.5). This consistency allows a

direct comparison between all methods presented in this dissertation. The evaluation of ECSPM is less extensive than that of CGSPM because they both use the same SPM link-prediction process. ECSPM only changed the spectral coarse-graining method. Therefore, many properties remain unchanged between ECSPM and CGSPM and the main goal of the experiments is to highlight their differences.

This section first defines the data, setup, and protocol used to conduct all experiments. Then, the link prediction accuracy and runtime results are described.

Terminology: The term "accuracy" is used informally and refers to the link prediction performance, that is, the ability to recover the hidden edges in a graph. The *accuracy metric* as defined by Equation (36) is not used in this evaluation.

6.5.1 Data Description

All datasets used in this evaluation are so-called "real world" graphs obtained from observations of interactions in practical applications, for example, social networks, collaboration networks, and e-mail communications. Most graphs are the same as in the CGSPM experiments. Additionally, the significantly larger *facebook*, *router*, *wiki*, and *enron* graphs are evaluated. All data preparation steps are described in Appendix B and graph statistics and data sources are listed in Table B.1.

6.5.2 Evaluation Protocol and Infrastructure

The experiments are conducted exactly as described in Section 5.5.2, that means, the evaluation process and infrastructure is exactly the same as in the CGSPM experiments. The link-prediction problem is interpreted as a binary classification problem. ECSPM defines a new classifier model implemented equally as the CGSPM model (see Algorithm 5.2) except for the GETPROJECTORS function which is implemented according to Algorithm 6.1. The perturbation set mapping is unchanged from CGSPM (see Section 5.3.1, Equation 68).

As in the CGSPM evaluation, each test produces a single result. A test is defined as a unique combination of classifier model, parameters, and graph. Each test is repeated 50 times for the graphs *florida*, *jazz*, *neural*, *USAir*, *netscience*, *metabolic*, *email*, *hamster*, and *yeast*. Every test involving the the graphs *facebook*, *router*, and *wiki* is repeated 15 times and tests on the *enron* graph are repeated only 3 times. All results are reported as the average value over all test repetitions. The detailed evaluation protocol, infrastructure and result formats are described in Section 5.5.2. Accuracy is evaluated by AUC, precision, and ROC curves as defined in Section 5.5.4. Runtime is measured as wall clock time elapsed from test start to test end.

Parameters Values: The SPM parameters p and s are set to the same values as in Lü et al. (2015): $p = 0.1$ and $s = 10$. The maximum number of iterations of Algorithm 6.2 is set to 20. The values for parameters k and b are defined below and their effect and semantics are described in Section 6.3.4. The remaining ECSPM parameters are set such that $e = k$, $z = k$, and $d = b$.

The effect of k on accuracy and algorithm runtime is thoroughly investigated in CGSPM. Since there is no principal change in ECSPM regarding the link prediction process, the parameter space of k is not investigated anew. With ECSPM many more eigenpairs can be preserved than with CGSPM. A hypothesis is that this allows to preserve high link prediction accuracy even when k is relatively small. Exploratory experiments have shown that a value for k in the interval $[50, 200]$ approximates and sometimes surpasses the AUC os SPM link prediction. The values 50, 100, and 200 were selected for a robust evaluation of this interval.

The optimal value for b is not trivial to know a priori because it depends on the unknown eigenvalue distribution around λ_e (λ_k). In Paratte and Martin (2016), a value of 500 is used while Di Napoli et al. (2016) uses much smaller values between 50 and 120. In Tremblay et al. (2016b) a value of 50 is used as well. Initial experiments suggest that a value for b of at least 70 is required for an accurate eigenvalue estimation. The values 70, 150, and 250 have been used in the experiments.

6.5.3 ECSPM Link Prediction Accuracy Results

Table 6.1 reports the link prediction AUC and precision results of ECSPM and SPM. The size of the *florida* and *jazz* graphs is smaller than $N = 200$, therefore, the experiments contain no results for $k = 200$ on these graphs. Furthermore, the evaluation of SPM on the *enron* graph was canceled after 48 hours of computation time without results. The best values in each row are highlighted in boldface.

ECSPM cannot reach the SPM AUC score with any of the chosen parameters on the three smallest graphs. However, on graphs larger than *neural*, ECSPM demonstrates higher or equal AUC than SPM with exception of the *hamster* graph. The *facebook* graph is the only graph where the best achieved AUC of ECSPM and SPM is equal. The standard deviation is low for all accuracy results.

No obvious pattern associates the parameters b and k to a particular trend in AUC score although a weak relationship can be seen for parameter k. The columns associated to $b = 70$ and $b = 250$ in Table 6.1 show the effect of increasing k with fixed b. Generally, a larger k increases the AUC score. This is expected because the preservation of more eigenspaces is hypothesized to increase accuracy. However, there are many exceptions such as the AUC of the *router* graph with parameters $b = 250/k = 50$ which is significantly better than with $b = 250/k = 100$. Furthermore, the *neural*, *USAir*, and *netscience* graphs perform best with $b = 70/k = 100$; not with $b = 70/k = 200$. As a consequence of these exceptions, a clear pattern cannot be established and the influence of the parameters appears to be specific to individual graphs.

The lower half of Table 6.1 contains the precision metric results. ECSPM cannot reach the precision of SPM in any of the conducted experiments. However, the precision scores are better than those achieved by CGSPM (c.f. Table 5.4) when the parameter k is set to similar values. CGSPM starts to approximate the SPM precision value when $k > 0.8N$ but $k = 200$ is much lower than $0.8N$ on the majority of evaluated graphs. Therefore, the ECSPM precision results show a significant improvement over CGSPM although the level of SPM cannot be matched with any of the evaluated parameters.

Table 6.1: *ECSPM AUC and precision for different parameters. Bold cells indicate the best value per row.*

ROC-AUC Results

		$b = 70$		$b = 150$	$b = 250$		
Graph	N	$k = 100$	$k = 200$	$k = 200$	$k = 50$	$k = 100$	SPM
florida	128	0.801±0.02	N/A	N/A	0.716±0.02	0.802±0.02	**0.945±0.01**
jazz	198	0.936±0.01	N/A	N/A	0.911±0.01	0.932±0.01	**0.976±0.00**
neural	297	0.871±0.01	0.855±0.01	0.851±0.01	0.852±0.01	0.854±0.01	**0.889±0.01**
USAir	332	**0.954±0.01**	0.943±0.01	0.939±0.01	0.938±0.01	0.943±0.01	0.929±0.02
netscience	379	**0.970±0.01**	0.951±0.02	0.943±0.02	0.947±0.02	**0.970±0.01**	0.947±0.02
metabolic	453	0.938±0.01	0.939±0.01	**0.941±0.01**	0.914±0.01	0.937±0.01	0.929±0.01
email	1133	0.866±0.01	0.884±0.01	**0.885±0.01**	0.842±0.01	0.868±0.01	0.880±0.01
hamster	1788	0.918±0.01	0.941±0.00	0.940±0.00	0.895±0.01	0.921±0.01	**0.951±0.00**
yeast	2224	0.830±0.01	0.853±0.01	**0.856±0.01**	0.805±0.01	0.833±0.01	0.832±0.01
facebook	4039	0.988±0.00	0.989±0.00	**0.991±0.00**	0.981±0.00	0.987±0.00	**0.991±0.00**
router	5022	0.581±0.03	0.647±0.02	0.726±0.02	**0.767±0.02**	0.698±0.02	0.617±0.01
wiki	7066	0.963±0.00	**0.967±0.00**	0.966±0.00	0.951±0.00	0.954±0.00	0.943±0.00
enron	33696	0.864±0.01	0.898±0.01	0.871±0.00	0.898±0.00	**0.901±0.00**	N/A

Precision Results

		$b = 70$		$b = 150$	$b = 250$		
Graph	N	$k = 100$	$k = 200$	$k = 200$	$k = 50$	$k = 100$	SPM
florida	128	0.171±0.02	N/A	N/A	0.145±0.03	0.171±0.03	**0.545±0.02**
jazz	198	0.336±0.03	N/A	N/A	0.311±0.03	0.330±0.02	**0.651±0.03**
neural	297	0.106±0.02	0.071±0.02	0.070±0.02	0.107±0.02	0.079±0.02	**0.165±0.02**
USAir	332	0.208±0.02	0.186±0.02	0.211±0.03	0.278±0.03	0.167±0.02	**0.442±0.03**
netscience	379	0.345±0.04	0.309±0.04	0.298±0.04	0.306±0.04	0.317±0.04	**0.404±0.04**
metabolic	453	0.158±0.02	0.181±0.03	0.171±0.02	0.148±0.02	0.144±0.02	**0.341±0.03**
email	1133	0.134±0.01	0.127±0.01	0.116±0.01	0.122±0.01	0.132±0.01	**0.151±0.01**
hamster	1788	0.157±0.01	0.118±0.01	0.094±0.01	0.118±0.01	0.103±0.01	**0.520±0.01**
yeast	2224	0.146±0.01	0.142±0.01	0.134±0.01	0.133±0.01	0.137±0.01	**0.169±0.01**
facebook	4039	0.388±0.01	0.385±0.00	0.376±0.01	0.296±0.00	0.324±0.01	**0.442±0.00**
router	5022	0.142±0.01	0.101±0.01	0.059±0.01	0.105±0.02	0.120±0.01	**0.164±0.01**
wiki	7066	0.141±0.00	0.133±0.00	0.137±0.00	0.128±0.00	0.135±0.00	**0.183±0.00**
enron	33696	0.054±0.00	0.086±0.02	0.054±0.00	0.073±0.00	**0.107±0.00**	N/A

All values are reported in the format <mean> ± <standard deviation>.

A ROC comparison reveals more details about the link prediction performance. Figures 6.3 and 6.4 show ROC curves for selected graphs. Appendix D.8 contains supplementary plots for the other graphs. Most ROC curves for graphs larger than *jazz* are contained in the 95% confidence interval of the corresponding SPM curve. Furthermore, all graphs larger than *neural* have ROC curves with a very steep initial incline that follows the SPM curve very closely. The *netscience*, *yeast*, and *wiki* graphs are representative examples of these observations. The *router* graph is an exception where the ECSPM ROC curves behave very differently.

The parameter b does not appear to influence the ROC curves in any perceptible way in Figure 6.3. The ROC curves cluster by the k value. This observation also applies to

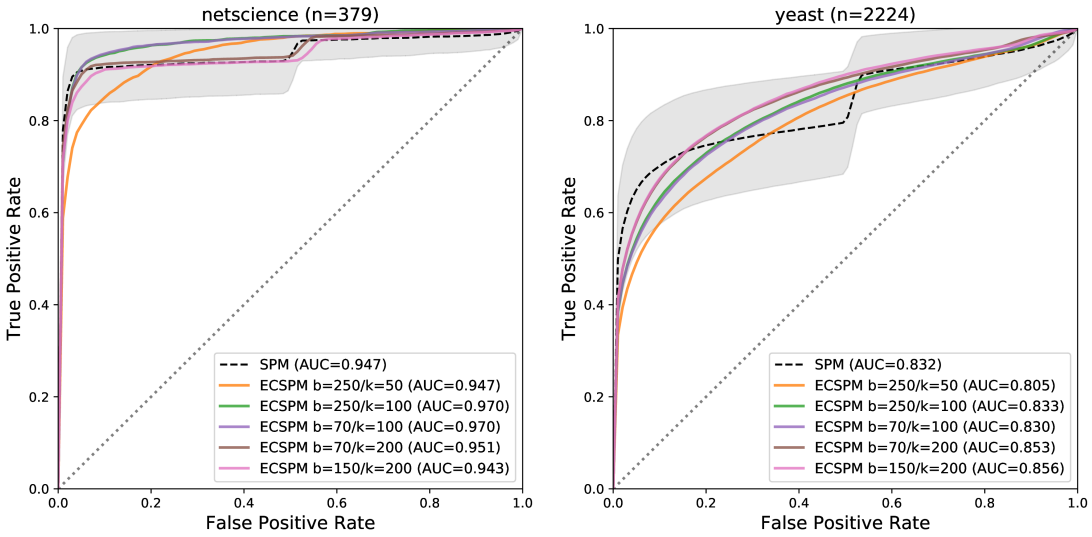


Fig. 6.3: *ECSPM example ROC curves for small and mid-sized graphs. The gray shaded area around the SPM curve represents the 95% confidence interval obtained by fitting a binomial distribution. Colored curves represent different parameter combinations.*

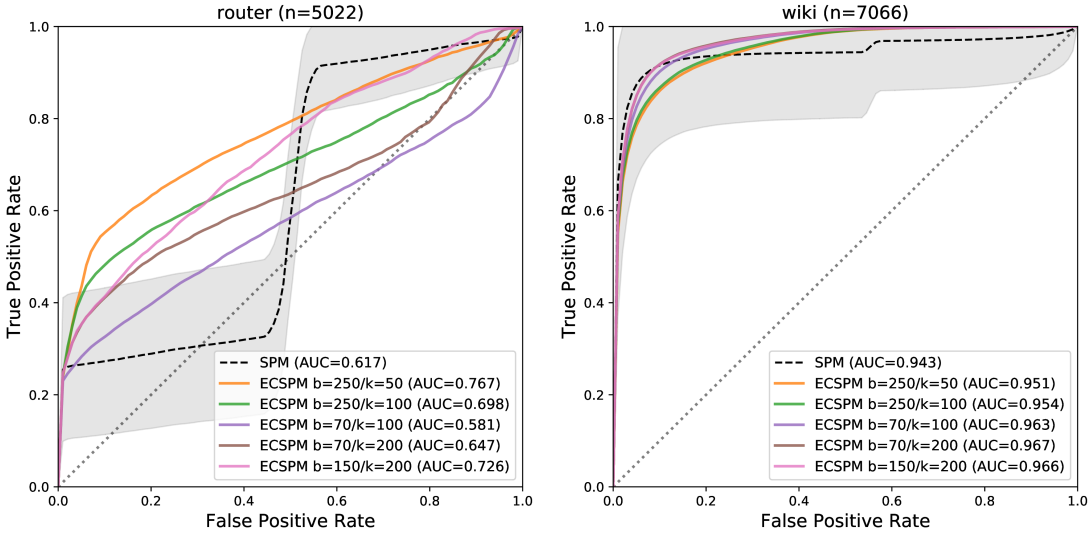


Fig. 6.4: *ECSPM example ROC curves for large graphs. The gray shaded area around the SPM curve represents the 95% confidence interval obtained by fitting a binomial distribution. Colored curves represent different parameter combinations.*

the other graphs with similar sizes, namely *netscience*, *metabolic*, *email*, and *hamster*. On smaller graphs most curves are overlapping closely and no clear difference is visible between either parameter b or k. The *wiki* (see Figure 6.4, right), *facebook*, and *enron* graphs also show no clear pattern regarding the parameters b and k. In contrast, the parameter b appears to have a significant influence on the *router* graph in Figure 6.4

(left). On this graph, the best parameter combination is $b = 250/k = 50$, that is, the largest polynomial order b and smallest k.

Figure 6.3 shows ROC curves for two smaller graphs. The *netscience* graph (left) displays the expected approximation behavior when k is close to N. This is shown by the $b = 150/k = 200$ and $b = 70/k = 200$ curves that copy the shape of the SPM curve very closely. This is evidence that the prediction characteristics are the same as those of SPM as k approaches N. However, perhaps surprisingly, the curves with $k = 100$ perform better while not approximating the characteristic shape of the SPM ROC curve very closely. The *yeast* graph (Figure 6.3, right) is almost one order of magnitude larger. A parameter value of $k = 200$ represents only a relative coarse-grained size of approximately $0.1N$ which can be considered very small. With relative sizes that are so small, the ROC curves do not approximate the SPM curve very closely. The unexpected observation is that several ROC curves have a higher AUC than SPM despite not approximating the SPM curve closely.

Figure 6.4 depicts the ROC curves for two significantly larger graphs. The *router* graph (left) is a special case that appears to be particularly difficult to predict. SPM only achieves an AUC of 0.617. The SPM ROC curves show that SPM systematically misclassifies edges after surpassing a TPR of 0.2 and the link prediction performance drops below the diagonal. Interestingly, ECSPM avoids this systematic error to a large extent and performs better than SPM. The *wiki* graph plot in Figure 6.4 (right) also shows ECSPM performing better than SPM despite $k = 200$ representing a relative coarse-grained graph size of less than $0.05N$. The *facebook* graph shows similar behavior as *wiki* but does not perform better than SPM.

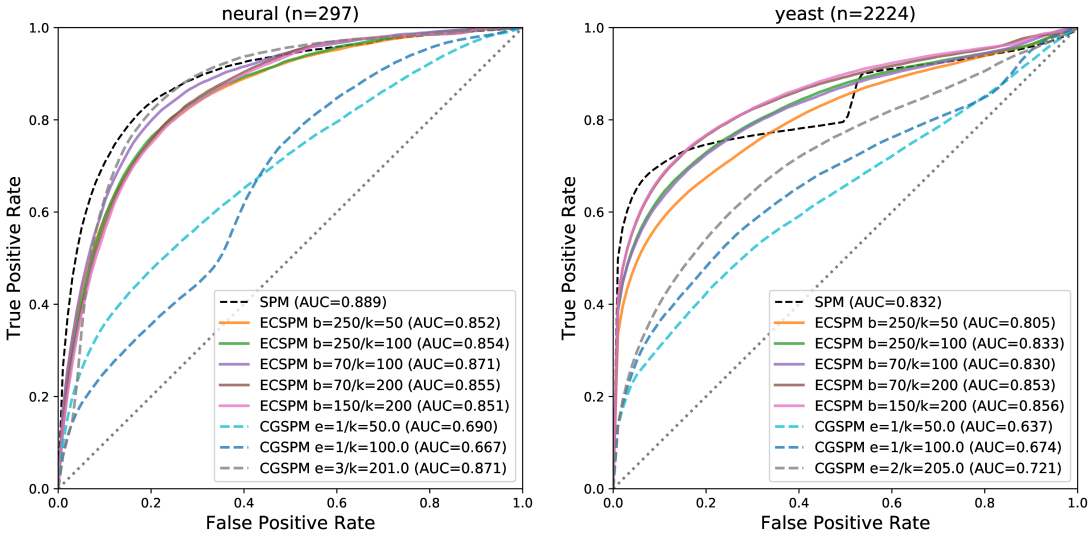


Fig. 6.5: *ECSPM (solid curves) compared to CGSPM (dashed curves) at same coarse-grained graph size.*

ROC Comparison with CGSPM: CGSPM can reach the SPM performance level in terms of accuracy metrics (see Table 5.4) and ECSPM can consistently surpass these levels (see Table 6.1). The most relevant design change between ECSPM and CGSPM is that the former preserves a large number of additional eigenpairs in the coarse-graining. The difference in accuracy between the two methods is likely due to this change. The ROC curves of both methods are compared at the same coarse-graining size k in order to investigate the effect of preserving $e = k$ eigenpairs versus $e \leq 10$ eigenpairs. Figure 6.5 shows the ROC curves of CGSPM and ECSPM. For each ECSPM parameter k the corresponding CGSPM curve is chosen by the cost-constrained strategy with lower bound $k-5$ and upper bound $k+5$. Therefore, the CGSPM curve represents the best AUC curve in the size interval $[k-5, k+5]$. Supplementary plots with all graphs results are shown in Appendix D.9.

On very small graphs, some CGSPM curves are performing better or equal to ECSPM as demonstrated in Figure 6.5 (left). The CGSPM curve for $e = 3/k = 201$ on the *neural* graph has the same AUC as the best ECSPM curve with parameters $b = 70/k = 200$. The *USAir* graph and all larger graphs show ECSPM performing significantly better and surpassing the AUC of the CGSPM curves. The gap between the ROC curves of different methods widens with the size of the original graph. On the *yeast* graph in Figure 6.5 (right), the CGSPM curves cannot reach the performance of ECSPM. The *yeast* graph is representative of the largest graphs in this comparison. These results show a clear advantage of preserving all k eigenpairs; and this advantage grows as the relative coarse-grained graph size k/N becomes smaller.

Table 6.2: *ECSPM link prediction runtime results in seconds. The best value on each graph is marked in boldface.*

		$b = 70$		$b = 150$	$b = 250$		
Graph	N	$k = 100$	$k = 200$	$k = 200$	$k = 50$	$k = 100$	SPM
florida	128	8.17±0.1	N/A	N/A	8.65±0.5	18.91±0.6	**0.59±0.0**
jazz	198	8.35±0.1	N/A	N/A	15.28±0.4	20.25±0.3	**0.92±0.0**
neural	297	9.97±0.1	19.09±0.1	16.04±0.3	18.18±0.4	30.30±0.7	**1.93±0.0**
USAir	332	10.22±0.1	18.62±0.1	16.86±0.2	18.60±0.6	36.21±1.2	**2.34±0.1**
netscience	379	9.72±0.1	19.40±0.1	18.05±0.6	27.32±1.6	24.62±0.8	**3.31±0.0**
metabolic	453	11.28±0.1	20.91±0.2	18.87±0.4	35.52±2.1	30.41±1.1	**5.46±0.0**
email	1133	**25.11±0.6**	32.91±0.7	33.61±0.4	56.92±1.6	98.32±2.1	69.62±1.3
hamster	1788	**41.37±0.4**	57.57±0.7	57.01±0.8	82.77±2.7	145.08±4.5	299.59±2.7
yeast	2224	**64.64±1.3**	81.03±2.3	86.53±1.4	203.53±5.5	269.23±12.3	1735.59±11.8
facebook	4039	**720.21±51.2**	768.89±21.1	1553.08±8.4	1708.03±49.2	2093.51±232	10403.28±148
router	5022	1287.63±12.5	**1113.16±172**	2665.56±5.6	3431.58±108	3548.09±12.9	10463.95±38.3
wiki	7066	2374.80±87.6	**1947.30±320**	4997.96±340	9083.60±29.5	9627.93±15.7	53181.87±329
enron	33696	**48894.93±386**	49517.82±1362	1.04e+05±0.0	1.27e+05±3936	1.06e+05±2217	N/A

All values are reported in the format <mean> ± <standard deviation>.

6.5.4 ECSPM Runtime Results

The runtime results measured in the ECSPM experiments are listed in Table 6.2. The runtimes for the $k = 200$ parameter for *florida* and *jazz* are missing because the graphs are too small. The SPM runtime for the *enron* graph is missing because the computation failed to complete within 48 hours. On small and mid-sized graphs, SPM is faster. However, on graphs larger than *metabolic*, ECSPM is faster. An interesting detail is the standard deviation of the runtime measurements and how if differs between columns. For example, the *router* graph has large variability for measurements with $k = 200$. Compare the corresponding columns for $b = 70$ (± 172) and $b = 150$ (± 5.6). The same columns of the *wiki* or *facebook* graphs show no significant difference.

Figure 6.6 plots the SPM and ECSPM runtimes as a function of graph size. Both axes of the plot are logarithmic and not scaled equally. The order of the polynomial approximation, that is, parameter b, influences the runtime strongly. The fastest ECSPM times use the $b = 70$ (solid purple and red curves) which is the smallest b parameter in the experiments. The next fastest runtime uses $b = 150$ (solid green curve) followed by $b = 250$ (solid orange and blue curves). While the parameter b is not as important for prediction accuracy, it is a dominant factor for runtime and computational cost.

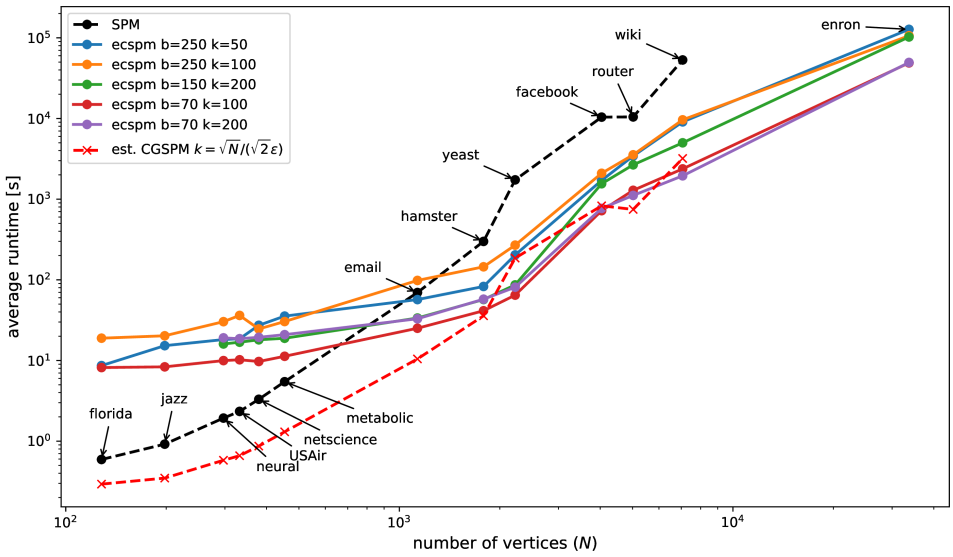


Fig. 6.6: *Runtimes of ECSPM, SPM, and CGSPM as a function of N for all graphs.*

Figure 6.6 shows converging runtimes for curves that share the same parameter b. The influence of k on the runtime diminishes as N becomes large. As expected from the theoretical analysis, ECSPM runtimes grow at a significantly lower rate than the SPM runtimes. Initially, ECSPM also grows significantly slower than the CGSPM runtime estimate (dashed red curve), but on larger graphs, the growths of the CGSPM estimate and ECSPM are similar.

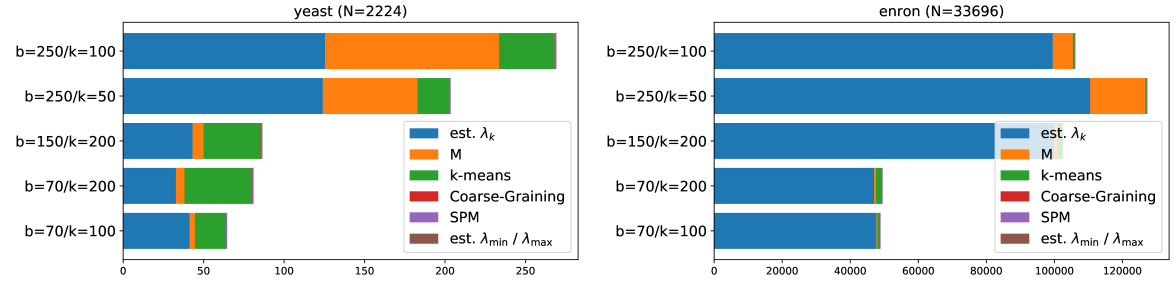


Fig. 6.7: *Runtime on different parts of the ECSPM link prediction process for the* yeast *and* enron *graphs. As the graph size increases, the runtime is dominated by the eigenvalue estimation.*

Figure 6.7 shows the time spent in different parts of the link-prediction process on two selected graphs. The left graphic is representative for mid-sized graphs and the right graphic reports the measurements on the largest graph. Supplementary graphics showing all graphs are shown in Appendix D.10. Most computation time is spent on the estimation of the eigenvalue λ_k (blue). Other notable factors are the computation of $\mathbf{M}$ (orange) and k-means clustering (green). Neither the coarse-graining projections nor SPM are a big factor and estimation of the extremal eigenvalues does not consume significant time either.

The bar charts highlight that the polynomial expansion filtering is currently the bottleneck. Most time is spent estimating λ_k and computing $\mathbf{M}$ which are both bounded by the complexity of the filtering operation. Comparing the *yeast* graph (left) to the *enron* graph (right) shows how the time to estimate λ_k start to dominates the runtime as the graphs grow large. It also shows that the parameter b is very influential for the runtime of the two most time-consuming steps. However, on the *yeast* graph, $b = 250/k = 50$ is faster than $b = 250/k = 100$ while the order is reversed on the *enron*. This means that other factors play an important role as well.

6.6 Discussion of Results

ECSPM is using, for the most part, the same link prediction process as CGSPM. Therefore, their link prediction characteristics are very similar. The difference is that ECSPM replaces the eigenspace partitioning method and implements a more accurate and efficient spectral coarse graining. Therefore, the experiments are less extensive and do not show all the properties that were investigated in the CGSPM evaluation. For example, based on a theoretical understanding of the coarse-graining mechanisms and the verification of this behavior in the evaluation of CGSPM, it is assumed that ECSPM approximates the SPM link prediction characteristics in the same way. However, this is not systematically evaluated. Instead, the experiments are designed to highlight the increased accuracy of ECSPM at lower values for k and to evaluate its algorithm runtime.

The partitioning approach used in ECSPM is k-means with k-means++ initialization. It is described in Section 4.3.4. Its major advantage is that the number of resulting partitions

can be controlled exactly. However, in de Lachapelle et al. (2008) this method is shown to offer less flexibility and lower accuracy guarantees than the fixed-size interval based approach that is used by CGSPM. Therefore, a secondary goal of this evaluation is the validation of the k-means eigenspace partitioning method.

This section follows the structure of the previous section that described the experimental results. First, link prediction accuracy results are discussed. Then, the the measured algorithm runtime is assessed in more detail.

6.6.1 Link Prediction Accuracy

The CGSPM evaluation established how AUC and precision scores gradually approach the SPM link prediction results and as k approaches N. In contrast, the ECSPM experiments evaluated only very few parameters combinations and did not conduct a parameter search. Nevertheless, this approximation behavior can be assumed to remain unchanged because it is a necessary consequence of the coarse-graining. This is explained in Section 5.6.2 and remains unchanged in ECSPM. The main argument is that, as k approaches N, the coarse-grained graph and the original graph start to become very similar and the link prediction characteristics must become similar as well. When $k = N$, both graphs are necessarily equal and the link prediction results will also be equal up to stochastic effects.

The largest value for k used in the ECSPM experiments is 200. On most evaluated graphs, this is much smaller than N. Therefore, most experiments can not reveal that ECSPM approximates the SPM results as the graph approximation is too small to reveal this effect. However, the *USAir* ($N = 332$) and *netscience* ($N = 379$) graphs are not much larger than $k = 200$. Appendix D.8 shows their ROC curve plots and it can be seen how the characteristic jumps in the ROC curves are approximated.

In summary, ECSPM necessarily exhibits the same approximation behavior as CGSPM when k approaches N because the link prediction process is the same. Some experimental evidence of this can be seen on smaller graphs but it is not systematically evaluated in the ECSPM evaluation.

Table 6.3: *Coarse-grained graph size and AUC comparison between ECSPM, SPM, and CGSPM at maximum ECSPM AUC score. Lowest k/N and highest AUC are highlighted in boldface.*

	florida	jazz	neural	USAir	netsci.	metab.	email	hamster	yeast	faceb.	router	wiki	enron
N	128	198	297	332	379	453	1133	1788	2224	4039	5022	7066	33696
SPM AUC	**0.95**	**0.98**	**0.89**	0.93	0.95	0.93	0.88	**0.95**	0.83	**0.99**	0.62	0.94	N/A
ECSPM AUC	0.80	0.94	0.87	**0.95**	**0.97**	**0.94**	**0.89**	0.94	**0.86**	**0.99**	**0.77**	**0.97**	**0.90**
ECSPM k/N	0.78	**0.50**	**0.34**	**0.30**	**0.26**	**0.44**	**0.18**	**0.11**	**0.09**	**0.05**	**0.01**	**0.03**	**0.003**
CGSPM AUC	0.81	0.95	0.87	N/A	N/A	N/A	N/A	0.94	N/A	N/A	N/A	N/A	N/A
CGSPM k/N	**0.71**	0.74	0.68	N/A	N/A	N/A	N/A	0.61	N/A	N/A	N/A	N/A	N/A

The second row in Table 6.3 displays the average AUC scores of SPM on all graphs. Furthermore, the third row shows the best average AUC scores of ECSPM and the fourth

row lists the corresponding relative coarse-grained graph sizes. The two last rows list the accuracy-constrained CGSPM results with the lower bound set to the maximum AUC of ECSPM. In most cases, CGSPM does not reach the AUC of ECSPM. Therefore many CGSPM values are unavailable, that means, the maximal AUC of CGSPM is worse than that of ECSPM.

The comparison in Table 6.3 shows that the ECSPM experiments reach AUC scores that are at least as good as SPM and CGSPM and in many cases even better. This statement is only violated on the three smallest graphs. It is very likely that ECSPM parameters exist that can reach even better AUC because the experiments did not optimize the parameters. Furthermore, as the graph sizes get larger, the ratio k/N at the maximum measured AUC becomes very small. This is strong evidence that k can be chosen independently of N. Specifically, the AUC of SPM can be reached on all evaluated mid-sized and large graphs when k is chosen in the interval $[70, 200]$; irrespective of the size of the graph.

As mentioned above and shown in Table 6.3, ECSPM is significantly more accurate than CGSPM. Additionally, the ratio k/N at the maximal AUC of ECSPM is smaller than that of CGSPM. The observed ECSPM AUC is measured at $k/N \leq 0.5$ on almost all graphs. As the graph sizes grow, this ratio becomes at least an order of magnitude smaller. In comparison, Table 5.5 shows that CGSPM reaches the maximal AUC at $k/N \geq 0.63$. Furthermore, Figure 6.5 compares the ROC curves of ECSPM and CGSPM at comparable values for k where data for both methods exists and shows that ECSPM clearly outperforms CGSPM.

The ROC curves also reveal that the classification error amplification problem affects ECSPM less than CGSPM. This limitation of the coarse-grained approach is discussed in Section 5.7. Figures 6.3 and 6.4 show that the ROC curves rise very steeply without major jumps in FPR. Such jumps have been observed in the CGSPM experiments (see Figure 5.8). They indicate that multiple edges are misclassified because they are classified as a group instead of individually. Initially, ECSPM ROC curves follow the SPM curves very closely which indicates that the top-ranked edges are classified accurately. This significantly improves AUC and precision scores.

The observation that the SPM AUC can be consistently surpassed by ECSPM is surprising. It appears very likely that some eigenpairs capture interactions that can be considered noise or biased in some manner. The preserved eigenpairs are determined by the parameter k. Sometimes a larger parameter k affects accuracy positively, but there are also examples where this has an detrimental effect. The noisy eigenpair hypothesis can explain this behavior. Ideally, an appropriate choice of k cuts-off all detrimental eigenpairs. A bad choice of k retains many detrimental eigenpairs. This raises the question of the optimal cut-off eigenvalue λ_k (k) for the polynomial filtering. However, this is a complex problem that has not been investigated in the scope of this study.

The approximation error discussion in Section 6.4.3 explains the influence of parameter b on the precision of the filter function. A precise filter isolates only the chosen eigenspaces in $\mathbf{M}$. The necessary precision depends on the density of eigenvalues surrounding λ_k. Therefore, parameters b and k are interdependent. Intuitively, a larger value should not

be harmful to accuracy. A low value means that contributions of unwanted eigenspaces are mixed into $\mathbf{M}$. On the other hand, the influence of unwanted, non-detrimental eigenspaces may not be very harmful to the coarse-graining. Overall, the parameter k appears to be more relevant for link prediction accuracy on most evaluated graphs. The *router* graph is atypical in this regard. The experiments show that the choice of b significantly influences link prediction accuracy on the *router* graph. Apparently is is very sensitive to the accuracy of the graph filter.

The key insights gained from the evaluation of link prediction accuracy can be summarized as follows.

- The preservation of a large number of eigenspaces is improving link prediction accuracy considerably.
- The use of k-means with k-means++ initialization for eigenspace partitioning can be fully validated. Having control over k outweighs any negative impact even if this method is less accurate than the fixed-size partitioning in CGSPM.
- The accuracy of ECSPM in terms of AUC scores is at least as good as SPM. On large graphs, it can be significantly better even though the relative coarse-grained graph size, that is, the ratio k/N, is very small. However, the precision scores at small k/N do not match the SPM precision.
- As graph sizes increase, the AUC and precision scores of ECSPM become significantly better than those of CGSPM even when the coarse-grained graph is at least an order of magnitude smaller than that of CGSPM.
- The experimental evidence suggests that some eigenpairs are detrimental to link prediction accuracy. The polynomial filtering approach used by ECSPM can remove such eigenpairs and improve prediction accuracy.
- The influence of parameters b and k on link prediction accuracy exhibits no clear pattern. The importance of either parameter depends strongly on the spectral features of each graph. Experimental results suggest that the choice of an appropriate value for k is generally more important than the choice of b.

6.6.2 Link Prediction Runtime

Based on the analysis in Sections 5.4.3 and 6.4.6, the computational cost and complexity of ECSPM on sparse graphs is estimated to be very similar to that of CGSPM. However, the accuracy evaluation has shown that ECSPM is more accurate at much smaller values for k which is an important factor for the computational cost of both methods. Additionally, ECSPM allows to choose k independently of the size of the original graph. Therefore, in practice, ECSPM should be significantly more efficient than CGSPM in terms of computational cost. The runtimes measured in the experiments on large graphs confirm this assumption.

In order to interpret the measured algorithm runtimes, the following context is important. The ECSPM implementation described in Section 6.4.4 is not optimized nor does it exploit any opportunities for parallelism. On the other hand, CGSPM and SPM use

very optimized numerical solvers to compute the eigenpairs of $\mathbf{A}$. On small scale problems, the runtime differences do not accurately reflect the computational cost because state-of-the-art numerical solvers are much more optimized than Algorithms 6.2 and 6.3.

Figure 6.6 compares runtimes for all three variants; SPM, CGSPM, and ECSPM. On small graphs, the ECSPM runtimes are the slowest. As shown in Figure 6.7, the estimation of λ_k is very time consuming. Most likely the relatively large runtime of ECSPM in this region is due to lacking optimization of its algorithms. Nevertheless, the ECSPM runtime displays a shallower runtime growth and it eventually becomes much faster than SPM on larger graphs.

It is unclear if the runtime growth of ECSPM is significantly different from that of the CGSPM estimate. The latter has been shown to be accurate in Section 5.6.3. Theoretically, their complexity should be similar but due to the much lower values for k, ECSPM should be able to eventually outperform CGSPM despite the inefficiencies of its implementation. Where data for both methods is available, ECSPM does not clearly outperform CGSPM. However, ECSPM appears to have a shallower runtime growth overall. Furthermore, no CGSPM data is available for the *enron* graph because an adequate parameter combination could not be sampled within 48 hours of computation time. In comparison, ECSPM computes accurate results within 13 to 31 hours.

This reveals that the main advantage of ECSPM is the high accuracy achievable with low k. This translates into a runtime advantage on large graphs. As shown in Figure 6.7, the link predictions runtime is negligible in practice when k is low. The runtime is dominated by the estimation of λ_k (Algorithm 6.2). The complexity of this process is bound by $O(mbkN)$ (see Section 6.4.6), where m is the number of iterations required for convergence and b and k are parameters. Since m is limited to a small constant and b and k have been chosen constant and independent from N, the runtime of this process grows linearly in N. On the other hand, the complexity of the SCG projections is bound by $O(kN^2)$ which is much higher (see Section 5.4.3). Therefore, the SCG projections are the actual bottleneck as N grows very large. Then, the runtime of ECSPM outperforms CGSPM due to the much lower values for k. Furthermore, this highlights the importance of developing a more efficient implementation of Algorithm 6.2.

Another advantage of ECSPM over CGSPM is the dependence of the latter on the knowledge of good parameters. Optimal parameters for CGSPM are difficult and expensive to find. The inability to locate good CGSPM parameters in reasonable time for the *facebook*, *router*, *wiki*, and *enron* graphs highlights this problem. In contrast, the ECSPM evaluation found values for k that produce AUC scores at least as good as SPM using only a few arbitrary choices of parameter k.

Overall, the parameter b strongly influences the runtime of ECSPM. Figures 6.6 and 6.7 show that link prediction runtimes with the same parameter b have very similar average runtime on large graphs. The parameter k has a smaller influence. However, it has a significant impact on the variance of some runtimes. For example, the *router* graph displays a large standard deviation of $\pm 172s$ with $k = 200$ (see Table 6.2). In contrast, with $k = 100$ the runtime standard deviation is only $\pm 12.5s$ and $\pm 12.9s$. The cause for this difference

is likely rooted in the estimation of λ_k because this part of the process dominates the runtime (see Figure D.10).

The runtime of the λ_k estimation is affected by the number of iteration required to converge on λ_k. Using the example of the *router* graph introduced above, assume λ_{100} is well separated from neighboring eigenvalues in the spectrum of the adjacency matrix. In this case, the parameter value for b is less important because the λ_k is easy to locate in a few iterations; even when the polynomial filtering is not very precise. On the other hand, assume λ_{200} is located in a dense cluster of very similar eigenvalues. An imprecise filter is likely to mix in contributions from neighboring eigenvalues. Therefore, Algorithm 6.2 may not be able to reliably locate the eigenvalue. Every time the trace of the approximated graph filter is estimated, small errors can lead to a different result that throw off the search algorithm. As a result, the search algorithm may accidentally find a value for λ_k early or keep searching until the maximum number of iterations is reached. This leads to highly variable runtimes.

A larger value for b increases the accuracy of the polynomial filtering which helps to reliably locate eigenvalues that are not well separated from neighboring eigenvalues (see Section 6.4.3). While this leads to less variable runtimes, a large b also significantly increases the computational cost of the filtering and therefore the overall runtime.

The following points summarize the most important conclusions about the runtime performance observed in the ECSPM experiments.

- ECSPM is faster than SPM on large graphs and its runtime grows slower than that of SPM with increasing graph size.
- The runtime of ECSPM is similar to CGSPM on large graphs. However, on the largest evaluated graph, ECSPM is the only method that produces a useful result within a 48 hours computation time limit.
- ECSPM achieves high link prediction accuracy with small values for k. Furthermore, k does not depend on the graph size. This, translates directly to a significant runtime advantage that increases as graphs become larger.
- The simplified parameter choice in ECSPM makes the computationally expensive search for optimal parameters less important. Accurate results can be obtained without conducting an exhaustive parameter search. In practice, useful results can be obtained with significantly reduced computational effort.
- Keeping the parameter b low is important for the runtime performance. On the other hand, a too small b can cause large runtime variance because the polynomial approximation is not precise enough to reliably locate the eigenvalue λ_k.

6.7 Conclusions

The efficient and controllable structural perturbation method (ECSPM) proposed in this chapter improves the coarse-grained link prediction classifier further while retaining the ability to exploit beneficial trade-offs between prediction accuracy and computational cost.

CGSPM requires the knowledge of a subset of eigenvectors in order to compute a coarse-grained graph and the computation of these eigenvectors is computationally very expensive. Therefore, only very few leading eigenpairs can be preserved in CGSPM. This limitation negatively affects its prediction accuracy. Additionally, the coarse-graining in CGSPM can not shrink the graph by more than approximately half its size without degrading link prediction accuracy significantly. And finally, the partitioning method of CGSPM cannot control the size of the coarse-graining graph reliably. In order to find good trade-offs, it requires a computationally very expensive parameter search to find optimal parameters.

ECSPM extends the eigenspace partitioning of CGSPM to resolve the parameter choice problem. It employs k-means clustering to partition the graph vertices. A critical advantage of k-means is that it makes the coarse-grained graph size a controllable parameter without increasing the overall complexity of the coarse-grained link prediction approach. Additionally, ECSPM circumvents the computation of eigenvectors entirely. A combination of polynomial filtering and random projections is used to approximate a feature space that is equivalent to the vector space spanned by the leading eigenpairs of the input graph. Since this approach is significantly more efficient than an eigendecomposition, a larger number of eigenspaces can be preserved in the coarse-graining which increases the link prediction accuracy of ECSPM.

An experimental evaluation verified that ECSPM is computationally efficient and highly accurate. Its accuracy surpasses that of CGSPM and, on many graphs even the AUC of SPM. Its runtime is comparable to those of CGSPM and superior to SPM on large graphs. Furthermore, the largest graph in the evaluation could only be processed by ECSPM. Therefore, ECSPM can scale to much larger graphs than SPM which fulfills one of the main motivations for the work presented in this thesis.

6.7.1 Limitations

A major limitation of the ECSPM evaluation is that no exhaustive parameter search has been conducted. Therefore, nothing is known about the optimality of the parameters k and b selected for evaluation. Furthermore, ECSPM allows more fine grained control with additional parameters but this has not been explored in this study.

ECSPM is an extension to CGSPM. Therefore, most limitations described in Section 5.7.2 apply to ECSPM as well. In particular, any coarse-graining reduces the precision scores significantly below the level of SPM. However, ECSPM is superior to CGSPM in this aspect. Nevertheless, ECSPM, just like CGSPM, is only suitable for applications that can afford to increase the false positive link prediction rate. This is an inherent consequence of the underlying trade-off between accuracy and computational cost.

Additionally, ECSPM only offers a significant runtime advantage on graphs that are sparse. On dense graph matrices, the polynomial filtering has a high computational cost and offers no computational cost advantage over CGSPM.

Chapter 7

General Conclusions

This dissertation reviewed the structural perturbation method for link prediction (SPM) and spectral coarse-graining (SCG). New error analyses for both of these methods were presented. Then, a new, coarse-grained SPM link prediction method (CGSPM) was proposed and thoroughly evaluated in order to learn about the effect of coarse-graining on link prediction accuracy and the runtime of the SPM algorithm. And finally, SCG has been extended with a new and computationally more efficient eigenspace partitioning method. This extension has been used to enhance CGSPM and the resulting efficient and controllable SPM (ECSPM) approach improved the usability and accuracy of CGSPM without increasing the computational cost. On a more general level, the methods implemented in ECSPM can be seen as a framework that enables to efficiently shrink graphs while preserving their spectral properties.

In Section 7.1 the two main contributions presented in this thesis are reviewed and put in context of the research questions and hypotheses discussed in Section 1.2. Finally, Section 7.2 outlines directions of future work.

7.1 Review of Main Contributions

This section reviews how the research questions and hypotheses defined in Section 1.2 are addressed by the contributions presented in Chapters 5 and 6.

Research Question 1: *Is SPM link prediction on coarse-grained graphs a viable approach to significantly reduce the high computational cost of SPM link prediction?*

Two hypotheses were formulated and evaluated to test this research question.

Hypothesis 1.1: SPM link prediction on coarse-grained graphs can exploit beneficial trade-offs between link prediction accuracy and computational cost.

To answer this hypothesis, the two accuracy metrics used in the evaluations of CGSPM and ECSPM have to be distinguished. When accuracy is measured with ROC AUC, the existence of exploitable and beneficial trade-offs was demonstrated for the CGSPM method in Chapter 5 and even better trade-offs have been observed with the ECSPM method on large graphs in Chapter 6. In contrast, when accuracy is measured by the precision metric, defined as in Lü et al. (2015) as the precision among the 10% highest ranked unobserved edges, neither CGSPM nor ECSPM were able to produce precision results comparable to SPM at any significant computational cost saving.

This discrepancy between accuracy metrics means that a coarse-grained approach can well separate between missing links (true positives) and non-existing links (true negatives).

This is what the ROC characteristics show. However, the loss of information incurred by the coarse-graining has a negative effect on the precision among a small subset of edges. A detailed analysis of the results has revealed that prediction errors in coarse-grained approaches have a higher impact because groups of edges are classified as one instead of classifying each edge independently. Therefore, a single error may cause multiple misclassifications.

As a result, an application must be willing to accept a larger number of edges that are predicted erroneously in order to benefit from coarse-grained SPM methods. For example, an applications that predicts top 10 results in some sort of user interface may see behavior where, sometimes, this list contains a large amount of errors because a group of edges was misclassified. On the other hand, an application that uses the rankings of all edges as a feature in a machine learning pipeline can expect better results because, on average, the CGSPM and ECSPM link-existence estimators can separate missing edges well from non-existing edges.

Hypothesis 1.2: SPM link prediction on coarse-grained graphs reduces the time complexity of SPM link prediction.

The complexity analyses of CGSPM (see Chapter 5) and ECSPM (see Chapter 6) are based on floating point operation counts. In both cases it has been shown that their time complexity is $O(kN^2)$. The complexity of SPM is $O(N^3)$ (see Chapter 3). Therefore, the confirmation of Hypothesis 1.2 depends on the relationship between k and N.

In case of CGSPM, this relationship is not entirely clear. A theory developed in this thesis suggests that $k = O(\sqrt{N})$ is required to maintain high link-prediction accuracy. Experimental evidence supports this theory but it has to be considered a worst case, or upper bound. A lower bound could not be established in the scope of this work. However, superior runtime at beneficial trade-offs could be confirmed in the CGSPM experiments. In case of ECSPM, k can be chosen small and independent of N, that means, k can be considered constant and the time complexity is bound by $O(N^2)$.

In experiments, the runtime of ECSPM is superior to SPM on large graphs. ECSPM can process a graph containing tens of thousands of vertices is a few hours. Neither SPM nor CGSPM could process this graph within two days of computation time. In the case of CGSPM, the main problem is the determination of good parameters as an exhaustive parameter search is a complex problem. The parameter choice for ECSPM is significantly simpler and good results could be achieved without an extensive parameter search. Assuming close to optimal parameters are known, CGSPM is expected to exhibit a comparable runtime to ECSPM on large graphs. On small and mid-sized graphs, CGSPM is the fastest method that was evaluated in this thesis.

In summary, Research Question 1 can has been positively answered by the theoretical analyses and the evaluations of CGSPM and ECSPM. The coarse-grained link-prediction approach, on average, produces an accurate global ranking of unobserved edges at significantly reduced computational cost.

Research Question 2: *Can spectral coarse-graining (SCG) be defined without knowledge of the eigenvalues and eigenvectors of a graph?*

This research question explored methods that allow to avoid the computation of an eigendecomposition. The motivation is to compute more accurate coarse grainings without increase in computational cost in order to increase link prediction accuracy. These approaches were implemented in ECSPM which is defined and evaluated in Chapter 6. The following two hypotheses were tested.

Hypothesis 2.1: Polynomial expansion filtering and random projections can be used to compute a spectral vertex similarity index suitable for spectral coarse-graining without requiring an eigendecomposition and with lower computational cost.

This hypothesis proposed that SCG can be implemented without an eigendecomposition by combining two specific methods that were used in different contexts for similar purposes. A spectral coarse-graining can be computed with a polynomial approximation of a graph filter and by k-means clustering of random vectors projected onto a polynomial filtering of the original adjacency matrix. The equivalence of this approach with the the partitioning used in SCG, has been shown theoretically in Section 6.3. The equivalence even holds, with high probability, under approximation errors. The fact that ECSPM produces highly accurate results confirms this empirically.

Therefore, a spectral coarse graining projector for a sparse matrix can be computed with complexity $O(ebN)$ where e is the number of eigenspaces that need to be preserved and b is the order of the approximation; which can be taken as a constant (see Section 6.4.6). In comparison, the computation of an SCG projector has a complexity of $O(eN^2)$.

Hypothesis 2.2: Given that the coarse-grained graph size is equal, the preservation of a large number of eigenspaces in a spectral coarse-graining improves link prediction accuracy on coarse-grained graphs.

This hypothesis has been directly tested and confirmed in the experiments conducted to evaluate ECSPM. In fact, the preservation of a large number of eigenspaces dramatically improves the link prediction ROC AUC. In some cases, the accuracy is consistently better than that of SPM. This indicates that there is an optimal number of eigenspaces to preserve and that the preservation of some eigenspaces, particularly those with small associated eigenvalues, may be detrimental to link prediction accuracy. However, the optimal number of eigenvalues to preserve is not known and likely graph dependent. In general, the confirmation of this hypothesis validates the approach outlined by Research Question 2 and implemented in ECSPM.

7.2 Future Work

In the scope of this thesis, the focus has been on the development of algorithms with low complexity and computational cost in terms of floating point operations. The presented implementations do not exploit opportunities for parallel computation or advanced low-level optimizations although some optimizations have been discussed briefly.

Such optimizations do not lower the order of complexity in terms of number of operations which determines the behavior of algorithm runtime as the graph size increases

towards infinity. A reduction of complexity is critical and has to be prioritized because any algorithm of high complexity is doomed to become infeasible to compute no matter how many other optimizations are applied. After computational complexity has been reduced, it is desirable to consider other optimization strategies to improve runtime further. In practice, such optimizations can reduce runtime significantly and make a method much more attractive.

In particular, the cut-off eigenvalue estimation in ECSPM, that is, Algorithm 6.2, is currently a bottleneck. An improvement in this algorithm's efficiency has the potential to speed-up the runtime of ECSPM significantly on large graphs. Given enough computational resources, it can be parallelized and distributed using approaches similar to those proposed in Shuman et al. (2011) for distributed signal processing. The experiments in Chapter 6 have shown that there is great potential to improve the algorithm runtimes as the eigenvalue estimation dominates the measured link prediction runtimes.

In order to understand the limits of link prediction accuracy with ECSPM, a detailed parameter space analysis should be conducted as was done for CGSPM. The experiments suggest an optimal, graph-dependent parameter k exists that removes all eigenspaces that are detrimental to link prediction accuracy (see Section 6.6.1). A detailed evaluation of this relationship could establish strategies for an optimal parameters choice.

Furthermore, the proposed CGSPM and ECSPM methods are limited by the complexity of the SCG projections which have a complexity of $O(kN^2)$. Due to the block-constant nature of the SCG projections, only k different values need to be computed. The proposed implementation of ECSPM is not optimized for this circumstance and applies operations to all N matrix elements independently. Due to the vertex partitioning that induces the coarse-graining projector, the elements that have the same coarse-grained value are known in advance and the amount of operations required to compute their values, in theory, is only k instead of N. Such an optimization could remove the projection bottleneck and reduce the time complexity of ECSPM further. However, this thesis did not address this bottleneck because it has not been observed in any of the ECSPM experiments. Nevertheless it has been shown to exist in the computational cost analysis (see Section 6.4.5).

Finally, a major motivation for the coarse-graining approach has been the idea that it generalizes to a large class of graph algorithms. The coarse-graining shrinks the input graph. While some information is lost in this processes, the semantics of the original graph are largely preserved. In particular, the coarse-graining can not express any new semantics that were not present in the graph before. Hence, the coarse-grained graph can be used in place of the original graph with any algorithm that take the original graph as input.

This dissertation only explored the impact of the coarse-graining on the accuracy and time performance of SPM; a spectral link-prediction algorithm. The core of the SPM algorithm has not been modified for the coarse-graining approach. This supports the hypothesis that the approach can be easily adapted to many other methods as well. SPM relies on the accuracy of the eigenpairs of the graph adjacency matrix and likely, the same approach can be transferred to other algorithms that have similar requirements. Nevertheless, this has not been demonstrated in this thesis and future work should explore how different methods are affected by coarse-grained approaches.

Appendices

Appendix A

Notation for Algorithms

All algorithms presented in this work use 1-based indexing for vectors and matrices and the following notation conventions for matrix access:

$\mathbf{M}(i)$	The i-th *column* of matrix $\mathbf{M}$.
$\mathbf{M}(i, j)$	The component M_{ij} of matrix $\mathbf{M}$.

Function calls are named after corresponding functionality in the *numpy/scipy*[8] suite of scientific computation software for the Python programming language; except for A(G) that has no equivalent in *numpy*.

EMPTY(N, M)	Creates an empty $N \times M$ matrix.
ZEROS(N, M)	Creates a $N \times M$ matrix with all components set to 0.
UNIQUE(v)	Returns a vector of the sorted unique values in vector v and a vector containing the number of times each unique value occurs in v.
EIGH($\mathbf{M}$)	Computes the eigendecomposition of matrix $\mathbf{M}$. Returns a vector of the eigenvalues in ascending order repeated according to their multiplicity and a matrix consisting of the orthonormal column eigenvectors. This uses LAPACK routine SSYEVD (Anderson et al., 1999) to perform the computation.
EIGSH($\mathbf{M}, \mathbf{k}$)	Computes k eigenvalues with largest value and the corresponding eigenvectors of matrix $\mathbf{M}$. Returns a vector of the eigenvalues in ascending order repeated according to their multiplicity and a matrix consisting of the orthonormal column eigenvectors. This uses ARPACK routines SSEUPD or DSEUPD to perform the computation.
EIGSHVAL($\mathbf{M}$, v)	Same as EIGSH but returns only the eigenvalues at indices contained in vector v.
RANDOMCHOICE(S, y)	Returns a uniform random sample of size y from the set S without replacement.
RANDN(N, M)	Returns a $N \times M$ matrix with components chosen uniformly at random from the standard normal distribution, i.e. $N(0, 1)$.
K-MEANS($\mathbf{M}$, k, v)	Computes a k-means clustering of the data points in the matrix $\mathbf{M}$ into k clusters. When $\mathbf{M}$ has N rows, each column defines a data-point in N-dimensional space. I is an optional initial clustering given as a vector of length k. Returns a partitioning Γ that assigns a group α to each data point in $\mathbf{M}$.

[8] https://docs.scipy.org

$\mathrm{A}(G)$	Returns the adjacency matrix of graph G.

Comments are marked with a ▷ symbol and may reference corresponding equations if applicable. Most variable names are chosen according to corresponding symbols in those equations.

Appendix B

Datasets

Table B.1: *Graphs used in the experimental evaluations.*

Label	Description	Vertices (N)	Edges	Ref.
florida	Trophic dynamics in florida ecosystem.	128	2075	1∗
jazz	Jazz musician collaboration.	198	2742	2∗
neural	Neural network of C. elegans.	297	2148	3
USAir	Air transportation in the US.	332	2126	4†
netscience	Coauthorship of scientists.	379	914	5
metabolic	Metabolic network of C. elegans.	453	2040	6
email	University Email communication.	1133	5451	7∗
hamster	Friendships of hamsterster.com users.	1788	12476	8∗
yeast	Protein-protein interaction network.	2224	7049	9†
facebook	Social circles from Facebook.	4039	88234	10‡
router	Internet topology	5022	6258	11
wiki	Wikipedia adminship voting network.	7066	103663	12‡
enron	Email communication network from Enron.	33696	180811	13‡

1 http://konect.uni-koblenz.de/networks/foodweb-baywet (Ulanowicz et al., 1998)
2 http://konect.uni-koblenz.de/networks/arenas-jazz (Gleiser and Danon, 2003)
3 http://www-personal.umich.edu/~mejn/netdata/celegansneural.zip (Watts and Strogatz, 1998)
4 http://vlado.fmf.uni-lj.si/pub/networks/data/mix/USAir97.net (Batagelj and Mrvar, 2006)
5 http://www-personal.umich.edu/~mejn/netdata/netscience.zip (Newman, 2006)
6 http://linkprediction.org/index.php/link/download/182 (Duch and Arenas, 2005)
7 http://konect.uni-koblenz.de/networks/arenas-email (Guimerà et al., 2003)
8 http://konect.uni-koblenz.de/networks/petster-friendships-hamster
9 http://vlado.fmf.uni-lj.si/pub/networks/data/bio/Yeast/Yeast.htm (Bu et al., 2003)
10 https://snap.stanford.edu/data/ego-Facebook.html (Leskovec and Mcauley, 2012)
11 http://penchet.com/ratha/LR_Source_code.rar (Pech et al., 2017)
12 https://snap.stanford.edu/data/wiki-Vote.html (Leskovec et al., 2010a,b)
13 https://snap.stanford.edu/data/email-Enron.html (Klimt and Yang, 2004, Leskovec et al., 2009)
∗ Konect: The Koblenz Network Collection (Kunegis, 2013)
† Pajek Datasets (Batagelj and Mrvar, 2006)
‡ SNAP Datasets: Stanford Large Network Dataset Collection (Leskovec and Krevl, 2014)

Table B.1 lists all datasets used in the evaluations conducted in this thesis. Each entry is a graph representation of interactions observed in the real world and recorded by people, application, or sensors.

The *florida*, *jazz*, *neural*, *USAir*, *netscience*, *metabolic*, *email*, *hamster*, *yeast*, and *router* graphs are the same as those used in the evaluation of SPM in Lü et al. (2015). This allows a direct comparison of results. In order to demonstrate the performance of CGSPM and ECSPM on larger graphs, the *facebook*, *wiki*, and *enron* graphs were selected. A randomly sampled subgraph of the *enron* dataset has been used in the original

evaluation of SPM by Lü et al. as well. Unfortunately, this sample has not been published which makes it impossible to compare to it.

All graphs in this work are unweighted, simple graphs (undirected, without self-loops). Where the graph in the original dataset is not simple and unweighted, edge direction and weights are ignored. Multiple edges, and self-links are removed. Importantly, only the largest connected component found in the graph is extracted and evaluated. The statistics presented in Table B.1 correspond only to the vertices and edges included in the largest connected component. These steps are equal to the graph pre-processing conducted in Lü et al. (2015).

Appendix C

Supplementary Tables

C.1 CGSPM Evaluation

Table C.1: *Optimal accuracy-constrained CGSPM parameters for all graphs.*

	florida		jazz		neural		USAir		netscience		metabolic		email		hamster		yeast	
	$N = 128$		$N = 198$		$N = 297$		$N = 332$		$N = 379$		$N = 453$		$N = 1133$		$N = 1788$		$N = 2224$	
$\Delta\mu\%$	b	e	b	e	b	e	b	e	b	e	b	e	b	e	b	e	b	e
MIN	110	4	100	9	20	5	10	7	360	10	390	10	600	6	1030	10	50	7
≤ 5	10	4	10	5	10	4	10	2	190	4	20	4	170	2	40	3	70	3
≤ 10	10	4	10	4	10	4	10	2	30	3	20	3	130	2	70	2	20	3
≤ 15	30	2	20	2	10	3	20	1	20	3	40	2	100	2	50	2	20	2
≤ 20	10	3	20	2	10	3	20	1	30	2	30	2	10	4	30	2	20	2
≤ 30	20	2	10	3	10	3	10	1	20	2	30	1	20	2	100	1	20	2
≤ 50	10	2	10	2	30	1	10	1	20	1	20	1	30	1	50	1	50	1
MAX	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1

Values in the $\Delta\mu\%$ column define accuracy bands defined as in Table 5.2. For example: ≤ 10 means maximally 10% accuracy loss starting from the border of the confidence interval for $\mu_{\text{spm}} - \mu_{\text{cgspm}}$.

Table C.2: *Optimal cost-constrained CGSPM parameters for all graphs.*

	florida		jazz		neural		USAir		netscience		metabolic		email		hamster		yeast	
	$N = 128$		$N = 198$		$N = 297$		$N = 332$		$N = 379$		$N = 453$		$N = 1133$		$N = 1788$		$N = 2224$	
$\frac{k}{N}$	b	e	b	e	b	e	b	e	b	e	b	e	b	e	b	e	b	e
0.1	10	1	10	1	20	1	20	1	30	1	10	2	110	1	20	2	20	3
0.2	10	1	30	1	10	2	10	2	30	2	10	2	10	3	40	2	20	3
0.3	30	1	10	2	20	2	20	2	20	3	30	2	10	5	60	2	20	3
0.4	50	1	10	3	10	3	10	3	20	5	40	2	10	5	90	2	200	2
0.5	10	2	20	2	10	3	10	4	20	6	20	3	110	2	40	3	290	2
0.6	10	2	20	2	10	4	10	6	20	6	60	2	140	2	200	2	410	2
0.7	10	2	20	3	20	3	10	7	20	6	20	5	70	3	80	3	50	7
0.8	10	3	10	5	10	5	10	7	160	3	20	8	90	3	80	4	50	7
0.9	30	2	20	5	10	7	10	7	280	3	40	4	60	5	1440	4	50	7
1.0	110	4	100	9	20	5	10	7	360	10	390	10	600	6	1030	10	50	7

Table C.3: *CGSPM AUC and precision with accuracy-constrained parameter optimization. Bold cells indicate the first value where SPM performance is matched (rows at the top are better). Graphs are ordered by size from smallest to largest.*

ROC-AUC Results

$\Delta\mu$%	florida	jazz	neural	USAir	netsci.	metabolic	email	hamster	yeast
MAX	0.551±0.03	0.595±0.04	0.607±0.04	0.830±0.04	0.563±0.06	0.628±0.07	0.500±0.00	0.500±0.00	0.500±0.00
≤ 50	0.735±0.02	0.776±0.02	0.652±0.02	0.830±0.04	0.731±0.03	0.747±0.03	0.697±0.03	0.702±0.04	0.637±0.02
≤ 30	0.808±0.02	0.864±0.01	0.815±0.02	0.830±0.04	0.829±0.03	0.778±0.02	0.759±0.01	0.800±0.01	0.741±0.01
≤ 20	0.858±0.02	0.902±0.01	0.815±0.02	0.862±0.02	0.838±0.03	0.821±0.02	0.779±0.01	0.855±0.01	0.741±0.01
≤ 15	0.874±0.02	0.902±0.01	0.815±0.02	0.862±0.02	0.878±0.02	0.848±0.02	0.801±0.02	0.882±0.01	0.741±0.01
≤ 10	0.931±0.01	0.926±0.01	0.863±0.01	0.910±0.01	0.889±0.02	0.879±0.01	0.833±0.01	0.899±0.01	0.763±0.01
≤ 5	0.931±0.01	0.954±0.01	0.863±0.01	0.910±0.01	0.913±0.03	0.902±0.01	0.852±0.01	0.925±0.01	0.786±0.01
MIN	**0.948±0.01**	**0.977±0.00**	**0.895±0.01**	**0.932±0.01**	0.938±0.02	**0.929±0.01**	**0.883±0.01**	**0.952±0.00**	0.807±0.01
SPM	0.945±0.01	0.976±0.00	0.892±0.01	0.924±0.01	**0.948±0.02**	0.926±0.01	0.880±0.01	0.951±0.01	**0.831±0.01**

Precision Results

$\Delta\mu$%	florida	jazz	neural	USAir	netsci.	metabolic	email	hamster	yeast
MAX	0.058±0.02	0.051±0.02	0.041±0.02	0.185±0.06	0.054±0.02	0.050±0.02	0.000±0.00	0.000±0.00	0.000±0.00
≤ 50	0.121±0.03	0.132±0.02	0.063±0.02	0.185±0.06	0.062±0.02	0.067±0.02	0.029±0.01	0.052±0.01	0.033±0.01
≤ 30	0.112±0.03	0.145±0.02	0.056±0.02	0.185±0.06	0.067±0.03	0.081±0.02	0.033±0.01	0.054±0.01	0.040±0.01
≤ 20	0.130±0.03	0.159±0.03	0.056±0.02	0.132±0.04	0.052±0.02	0.060±0.02	0.033±0.01	0.046±0.01	0.040±0.01
≤ 15	0.105±0.03	0.159±0.03	0.056±0.02	0.132±0.04	0.088±0.03	0.052±0.02	0.013±0.01	0.038±0.01	0.040±0.01
≤ 10	0.363±0.13	0.174±0.03	0.054±0.02	0.279±0.03	0.081±0.03	0.051±0.02	0.010±0.00	0.032±0.01	0.031±0.01
≤ 5	0.363±0.13	0.236±0.03	0.054±0.02	0.279±0.03	0.282±0.10	0.034±0.02	0.007±0.00	0.042±0.01	0.011±0.00
MIN	**0.549±0.02**	**0.652±0.02**	0.094±0.04	0.082±0.04	0.374±0.06	**0.348±0.03**	0.138±0.02	0.511±0.02	0.006±0.00
SPM	0.547±0.02	0.652±0.02	**0.166±0.02**	**0.441±0.03**	**0.409±0.05**	0.344±0.03	**0.148±0.01**	**0.521±0.01**	**0.171±0.01**

Values in the $\Delta\mu$% column define accuracy bands defined as in Table 5.2. For example: ≤ 10 means maximally 10% accuracy loss starting from the border of the confidence interval for $\mu_{\text{spm}} - \mu_{\text{cgspm}}$.
All values are reported in the format <mean> ± <standard deviation> for a sample of size 100.

Table C.4: *CGSPM AUC and precision with cost-constrained parameter optimization. Bold cells indicate the first value where SPM performance is matched (rows at the top are better). Graphs are ordered by size from smallest to largest.*

ROC-AUC Results

$\frac{k}{N}$	florida	jazz	neural	USAir	netscience	metabolic	email	hamster	yeast
0.1	0.551±0.03	0.595±0.04	0.637±0.03	0.862±0.02	0.751±0.03	0.794±0.02	0.725±0.01	0.846±0.01	0.763±0.01
0.2	0.551±0.03	0.656±0.02	0.712±0.02	0.910±0.01	0.838±0.03	0.794±0.02	0.759±0.02	0.868±0.01	0.763±0.01
0.3	0.559±0.03	0.776±0.02	0.734±0.02	0.917±0.01	0.878±0.02	0.821±0.02	0.780±0.01	0.892±0.01	0.763±0.01
0.4	0.658±0.03	0.864±0.01	0.815±0.02	0.922±0.01	0.894±0.02	0.848±0.02	0.780±0.01	0.912±0.01	0.769±0.01
0.5	0.735±0.02	0.902±0.01	0.815±0.02	**0.927±0.01**	0.899±0.02	0.879±0.01	0.816±0.01	0.925±0.01	0.790±0.01
0.6	0.735±0.02	0.902±0.01	0.863±0.01	0.930±0.01	0.899±0.02	0.880±0.02	0.838±0.01	0.936±0.01	0.801±0.01
0.7	0.735±0.02	0.934±0.01	0.871±0.01	0.932±0.01	0.899±0.02	0.908±0.01	0.859±0.01	0.946±0.01	0.807±0.01
0.8	0.858±0.02	0.954±0.01	0.884±0.01	0.932±0.01	0.901±0.03	0.914±0.01	0.872±0.01	0.950±0.00	0.807±0.01
0.9	0.874±0.02	0.967±0.01	**0.893±0.01**	0.932±0.01	0.918±0.03	0.916±0.01	**0.881±0.01**	**0.952±0.00**	0.807±0.01
1.0	**0.948±0.01**	**0.977±0.00**	0.895±0.01	0.932±0.01	0.938±0.02	**0.929±0.01**	0.883±0.01	0.952±0.00	0.807±0.01
SPM	0.945±0.01	0.976±0.00	0.892±0.01	0.924±0.01	**0.948±0.02**	0.926±0.01	0.880±0.01	0.951±0.01	**0.831±0.01**

Precision Results

$\frac{k}{N}$	florida	jazz	neural	USAir	netscience	metabolic	email	hamster	yeast
0.1	0.058±0.02	0.051±0.02	0.062±0.02	0.132±0.04	0.054±0.03	0.084±0.02	0.022±0.01	0.055±0.01	0.031±0.01
0.2	0.058±0.02	0.101±0.02	0.048±0.01	0.279±0.03	0.052±0.02	0.084±0.02	0.032±0.01	0.040±0.01	0.031±0.01
0.3	0.098±0.02	0.132±0.02	0.047±0.01	0.196±0.04	0.088±0.03	0.060±0.02	0.031±0.01	0.032±0.01	0.031±0.01
0.4	0.108±0.02	0.145±0.02	0.056±0.02	0.171±0.03	0.085±0.03	0.052±0.02	0.031±0.01	0.028±0.01	0.006±0.00
0.5	0.121±0.03	0.159±0.03	0.056±0.02	0.106±0.03	0.091±0.03	0.051±0.02	0.011±0.01	0.042±0.01	0.005±0.00
0.6	0.121±0.03	0.159±0.03	0.054±0.02	0.063±0.03	0.091±0.03	0.043±0.02	0.009±0.01	0.020±0.01	0.003±0.00
0.7	0.121±0.03	0.175±0.03	0.039±0.01	0.082±0.04	0.091±0.03	0.026±0.01	0.011±0.01	0.042±0.01	0.006±0.00
0.8	0.130±0.03	0.236±0.03	0.044±0.01	0.082±0.04	0.056±0.06	0.015±0.01	0.008±0.00	0.121±0.10	0.006±0.00
0.9	0.105±0.03	0.392±0.07	0.042±0.03	0.082±0.04	0.331±0.07	0.019±0.01	0.023±0.03	0.510±0.02	0.006±0.00
1.0	**0.549±0.02**	**0.652±0.02**	0.094±0.04	0.082±0.04	0.374±0.06	**0.348±0.03**	0.138±0.02	0.511±0.02	0.006±0.00
SPM	0.547±0.02	0.652±0.02	**0.166±0.02**	**0.441±0.03**	**0.409±0.05**	0.344±0.03	**0.148±0.01**	**0.521±0.01**	**0.171±0.01**

All values are reported in the format <mean> ± <standard deviation> for a sample of size 100.

Appendix D

Supplementary Figures

D.1 CGSPM: Coarse-Grained Matrix Size in Parameter Space

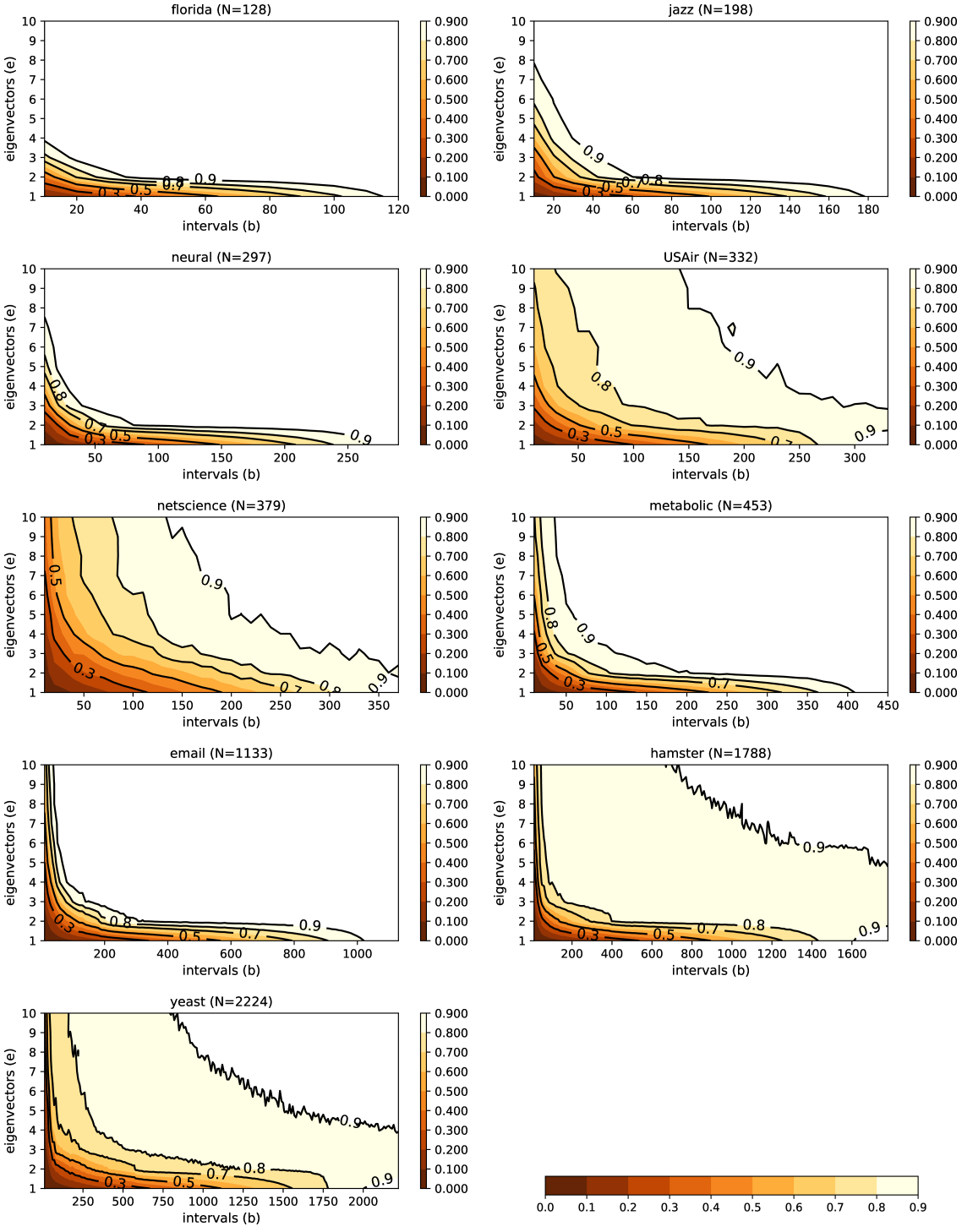


Fig. D.1: *Relative coarse-grained graph size k/N for the sampled section of the parameter space. Dark-red colors indicate big size reduction ($k > 0.3N$) and bright regions insignificant size reduction ($k \geq 0.9N$).*

D.2 CGSPM: AUC Difference in Parameters Space

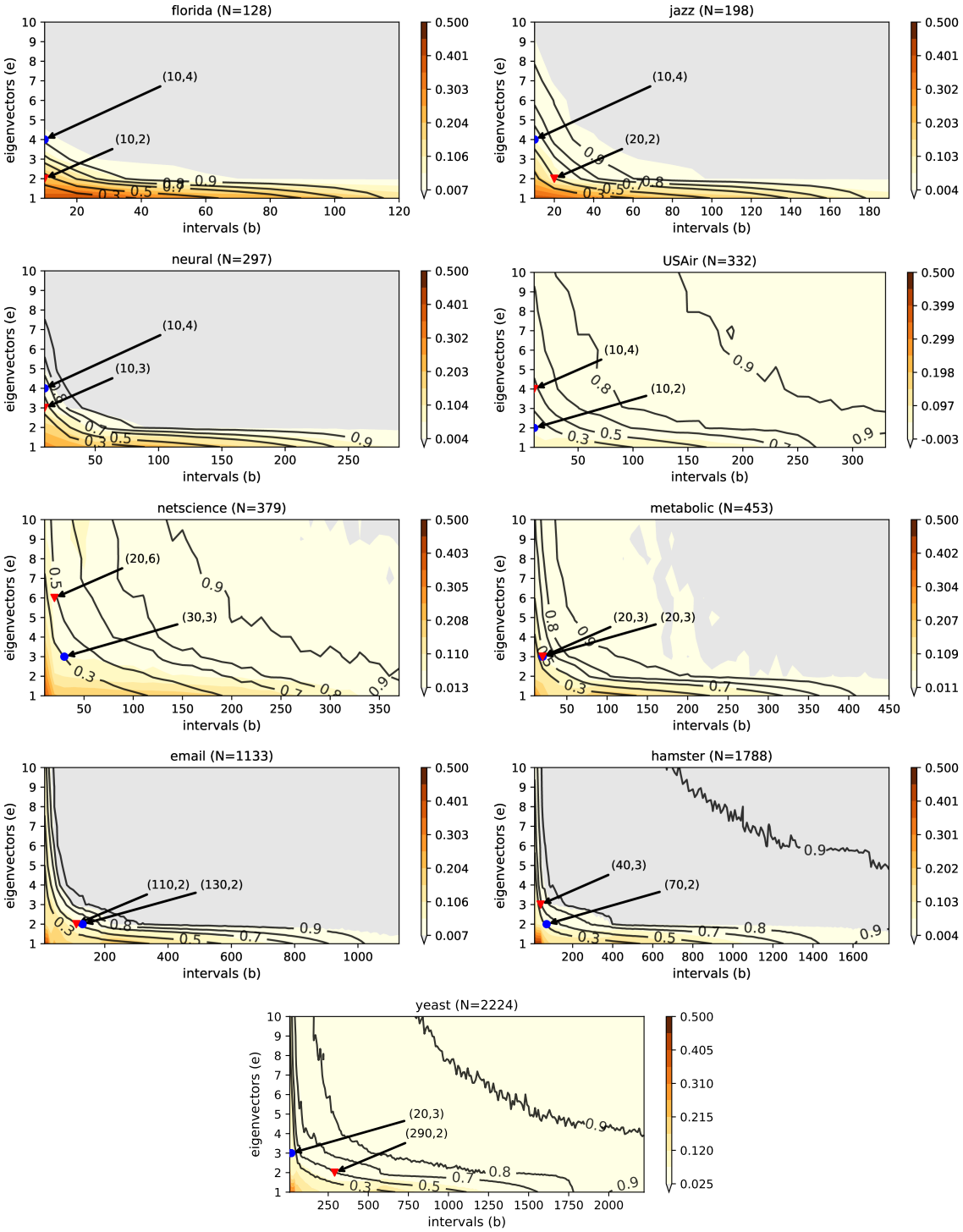


Fig. D.2: *Parameter space for AUC contrast. Dark indicates a large difference; bright and gray colors show a small difference (good performance). Black contours indicate relative coarse-grained graph size k/N which represents computational cost. Optimal accuracy (red dot) and cost (blue triangle) constraint parameters are annotated.*

D.3 CGSPM: AUC Plots

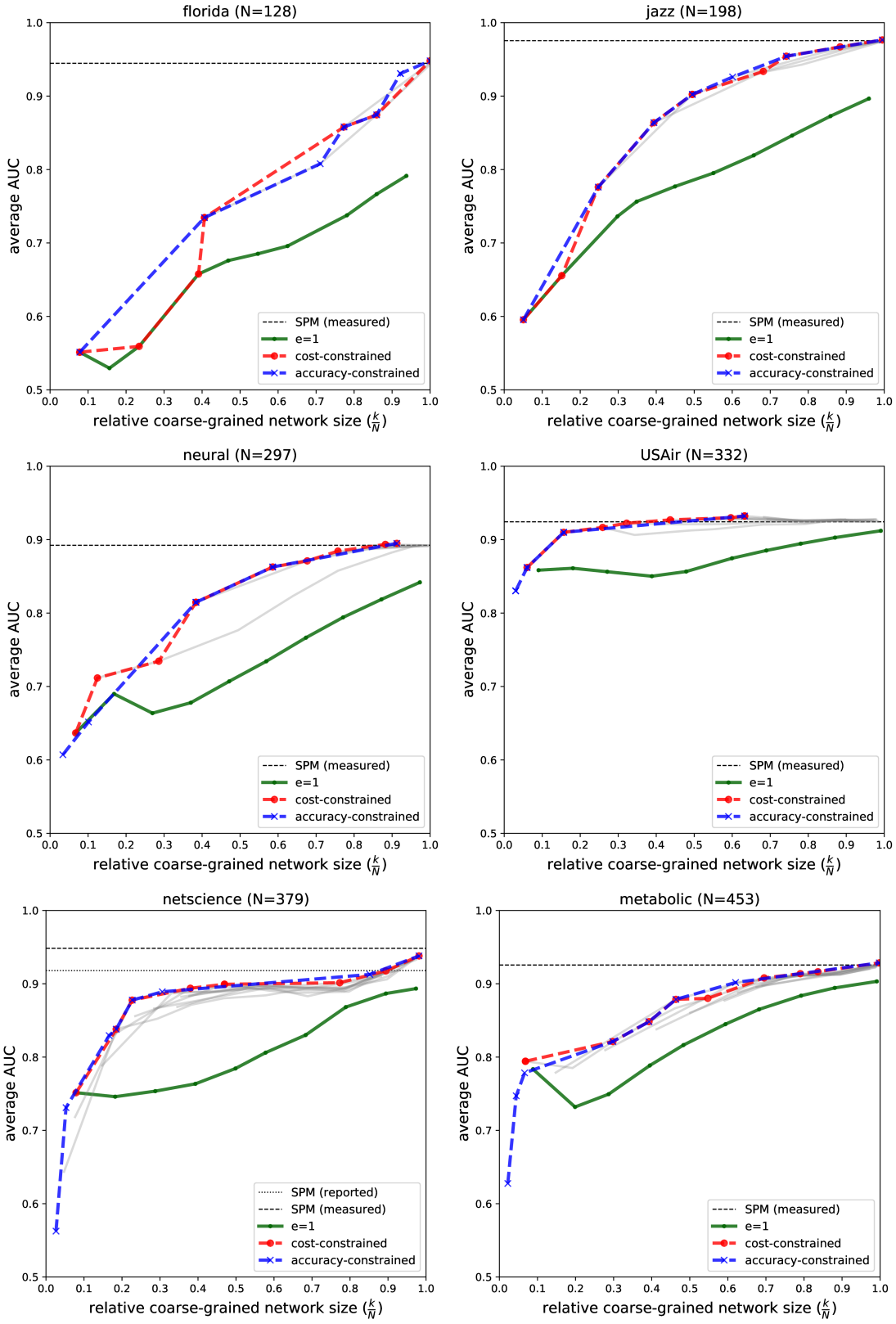


Fig. D.3

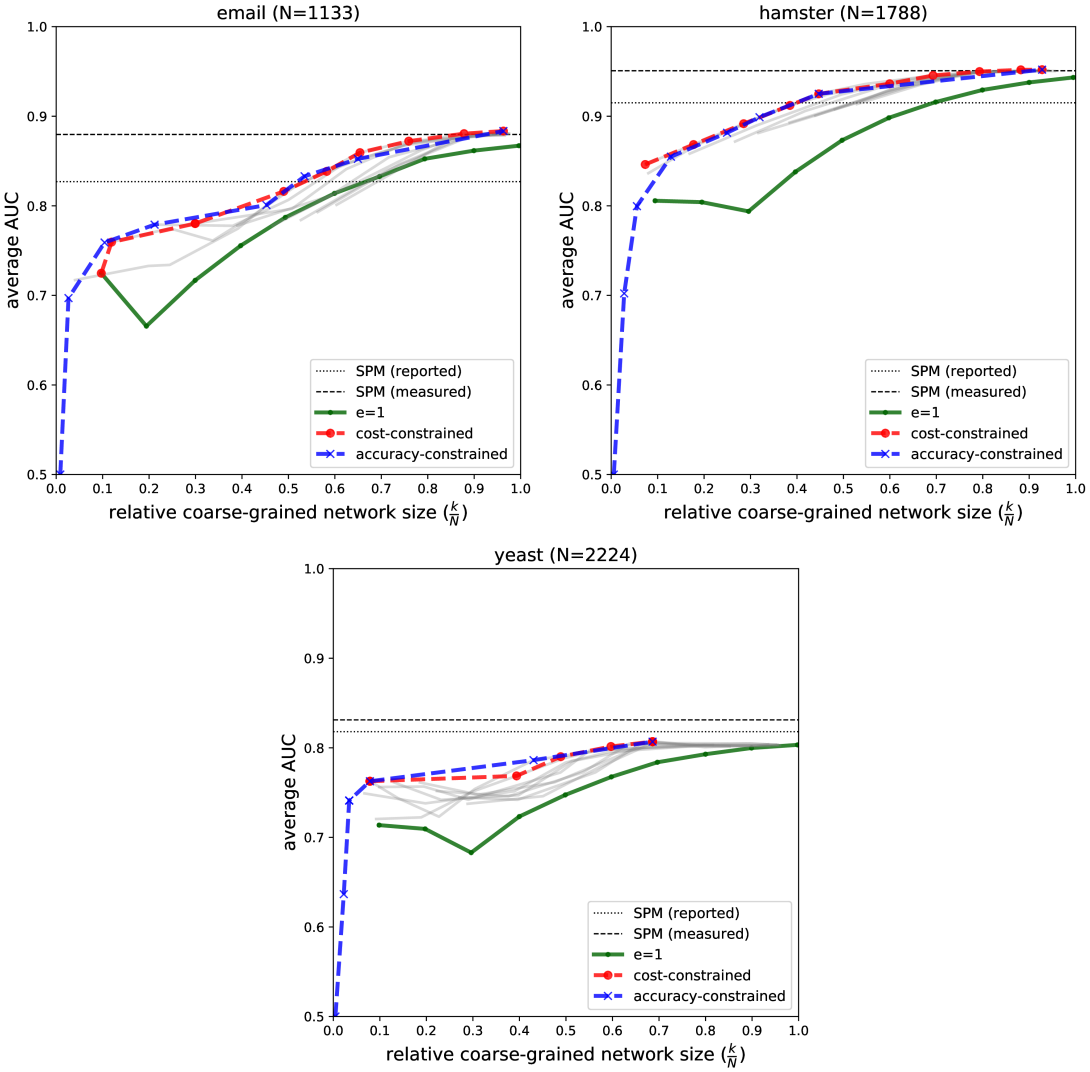


Fig. D.3 (cont.)**:** *Link prediction performance as a function of the relative coarse-grained network size* $\frac{k}{N}$*. The AUC level of SPM measured in the evaluation is indicated as a black dashed horizontal line. The reported AUC in Lü et al. (2015) is shown with a black horizontal dotted line if it was obtained on a graph sample. The accuracy-constrained (blue) and cost-constrained (red) parameter optimization strategies are highlighted. The green curve shows AUC for* $e = 1$ *when varying the b parameter. The gray curves trace the AUC using the same parameter choice strategy for* $e > 1$*.*

D.4 CGSPM: Precision Plots

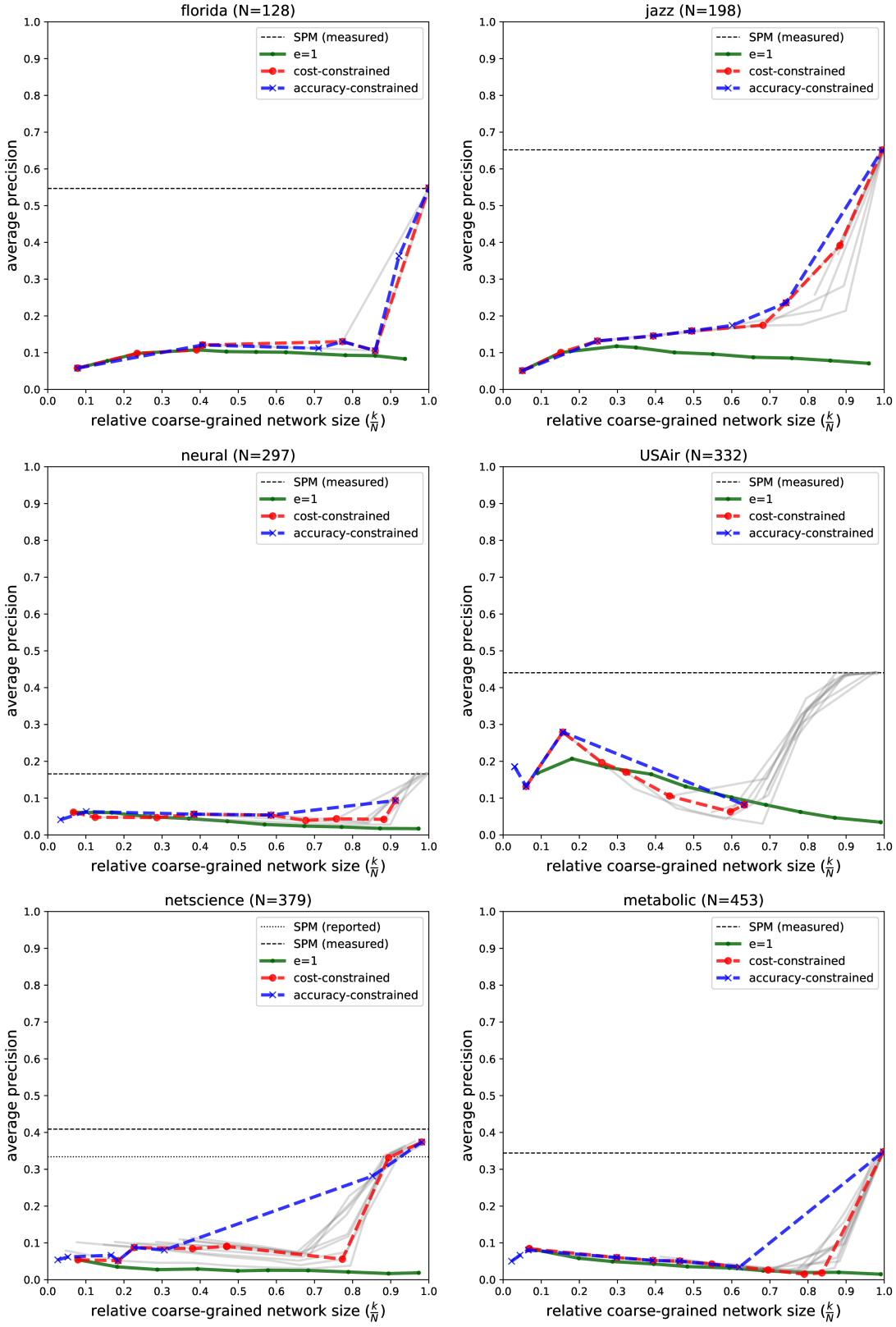


Fig. D.4

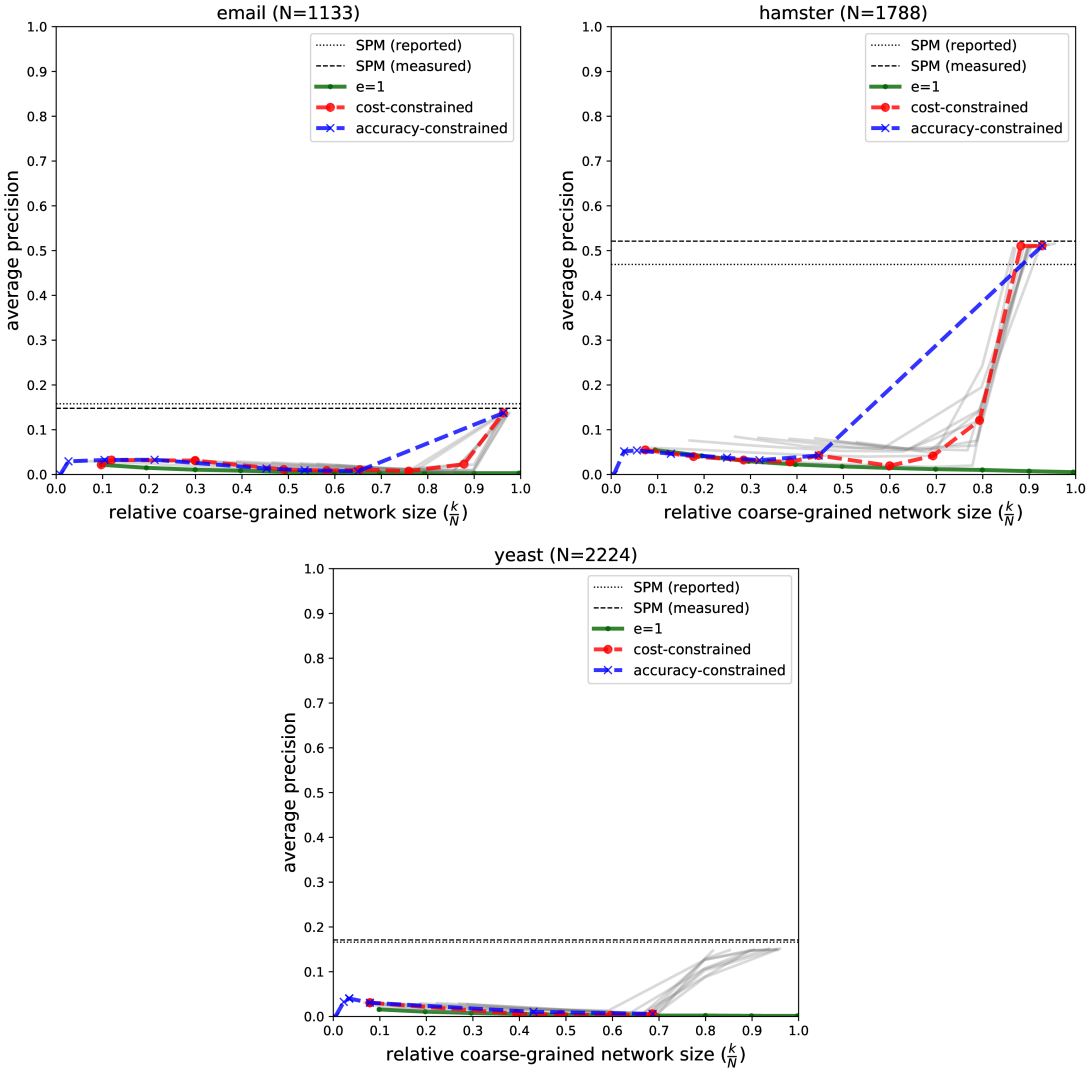


Fig. D.4 (cont.)**:** *Link prediction performance as a function of the relative coarse-grained network size* $\frac{k}{N}$*. The precision level of SPM measured in the evaluation is indicated as a black dashed horizontal line. The reported precision in Lü et al. (2015) is shown with a black horizontal dotted line if it was obtained on a graph sample. The accuracy-constrained (blue) and cost-constrained (red) parameter optimization strategies are highlighted. The brown curve shows precision for* $e = 1$ *when varying the* b *parameter. The gray curves trace the precision using the same parameter choice strategy for* $e > 1$*.*

D.5 CGSPM: Averaged ROC Curve Plots

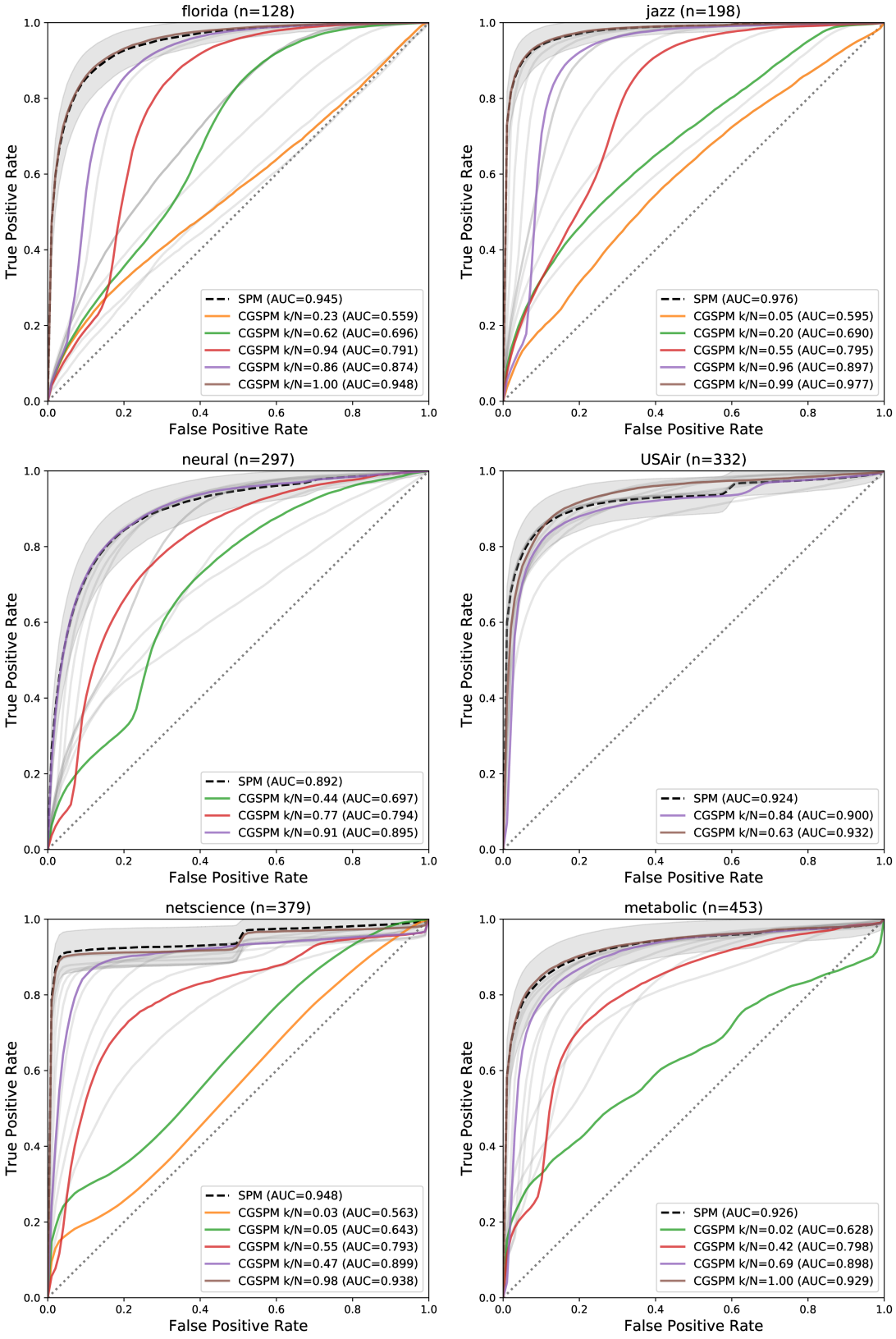


Fig. D.5

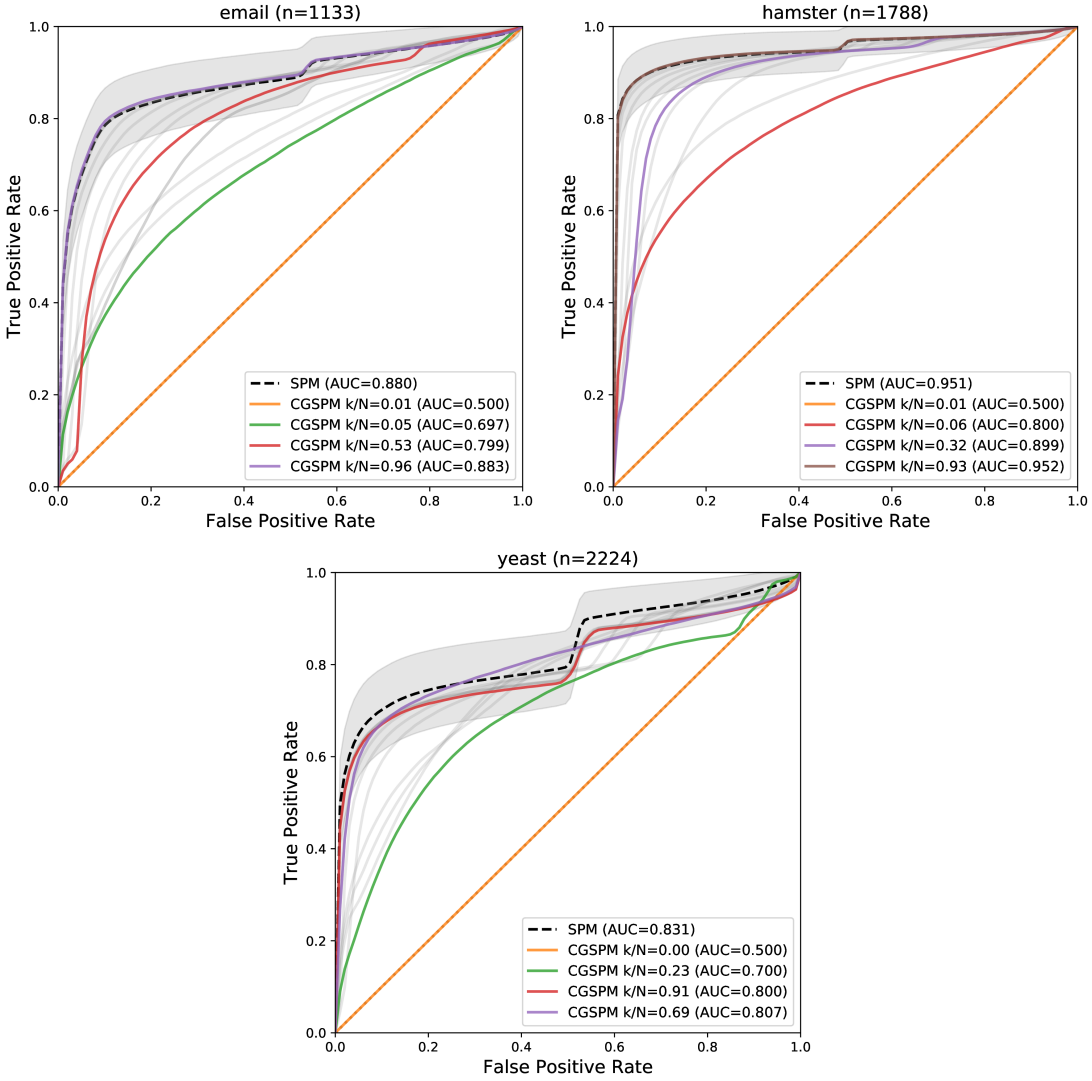


Fig. D.5 (cont.)**:** *Averaged ROC Curves for all evaluated graphs. Different curves in the same plot show CGSPM performance at different parameter combinations. Gray curves indicate the distribution of the bulk of ROC curves. The best curve in each AUC intervals of* $[0.5, 0.6]$, $[0.6, 0.7]$, $[0.7, 0.8]$, *and* $[0.8, 0.9]$ *is highlighted. In certain graphs, various intervals can be empty but the same color indicates the same interval in each graph. The shaded area surrounding the averaged SPM curve indicates a 95% confidence interval based on fitting a binomial distribution.*

D.6 Mapped vs. Projected CGSPM: Averaged ROC Curve Plots

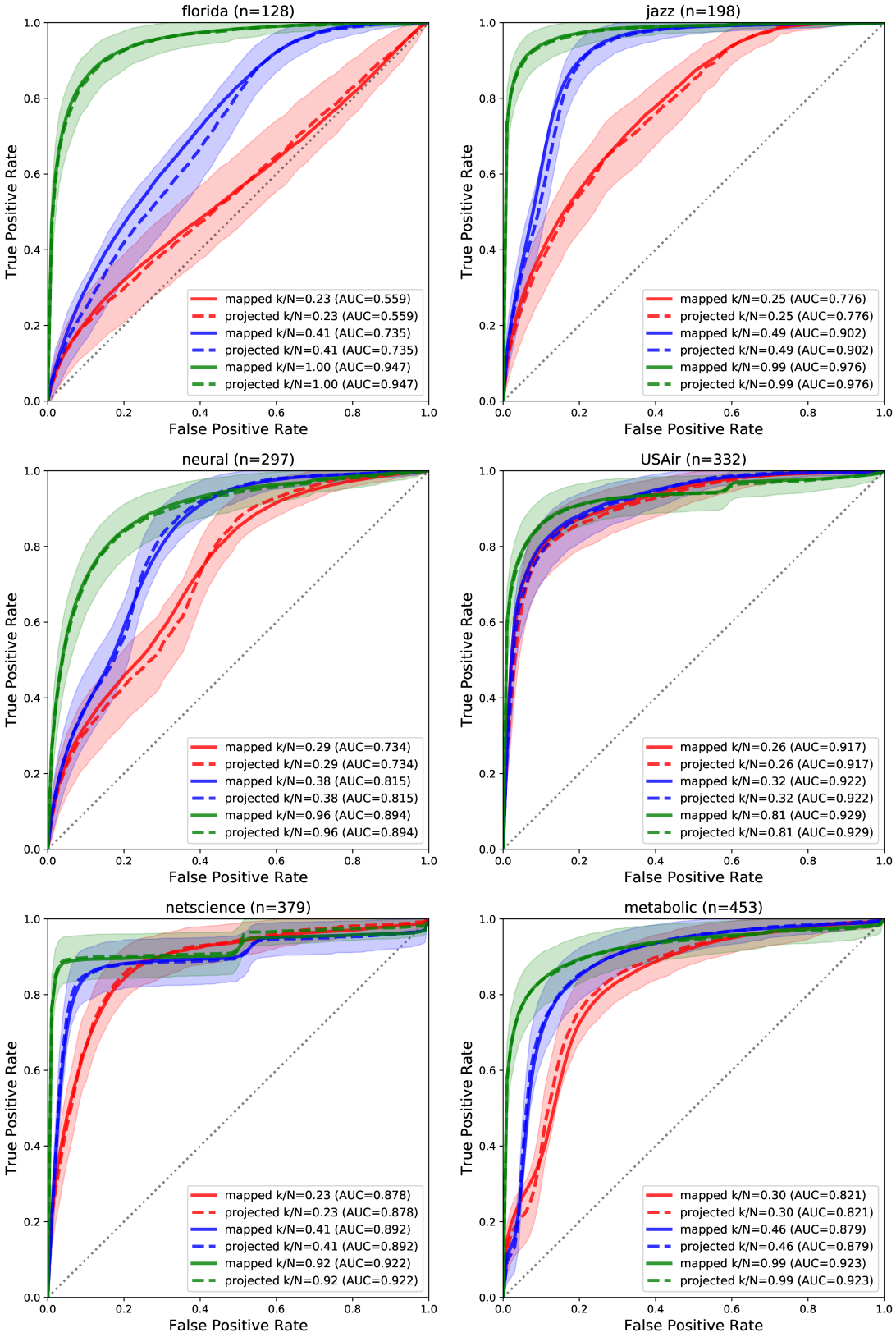


Fig. D.6

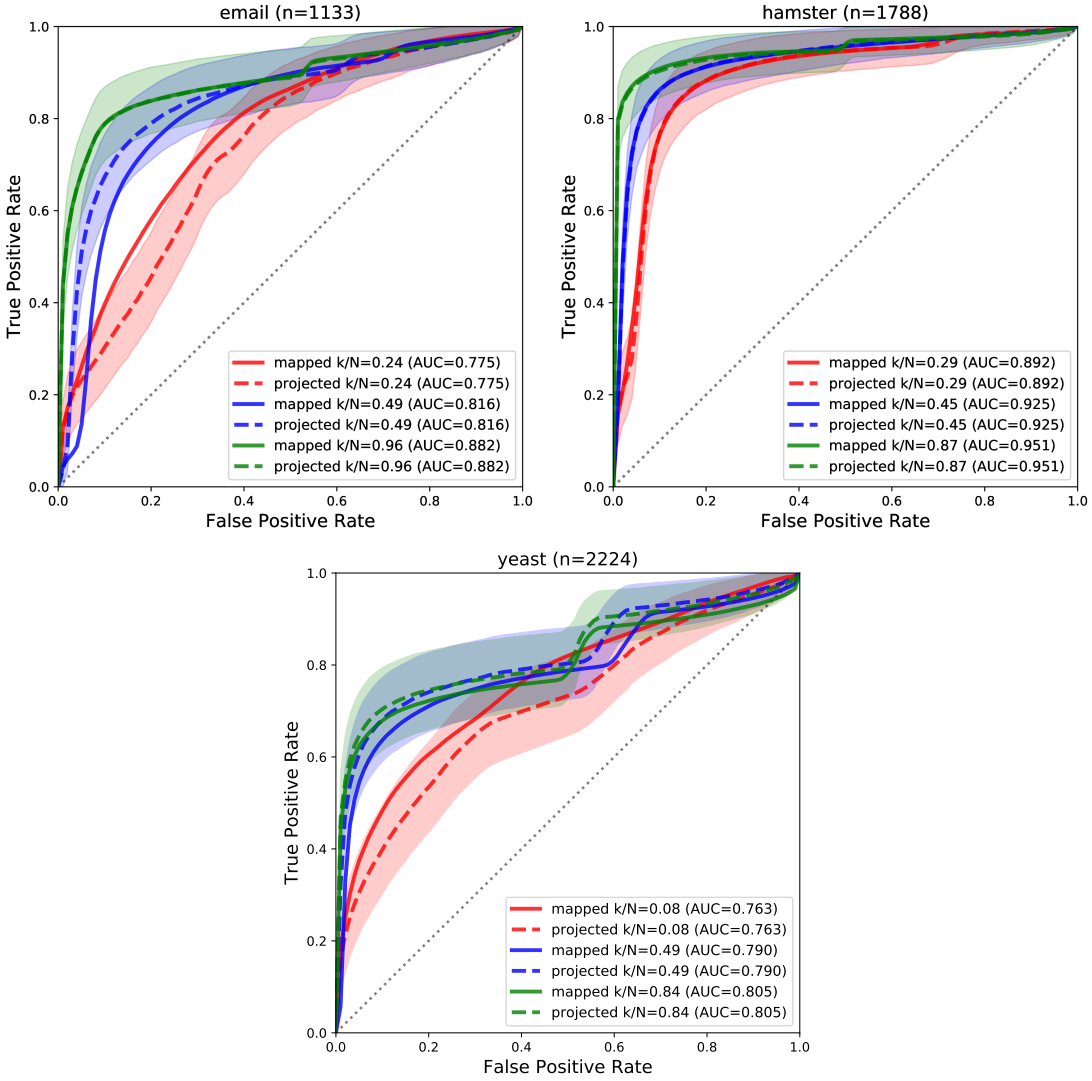


Fig. D.6 (cont.)**:** *ROC Curves for CGSPM with projected and mapped perturbation set. The solid curve is the mapped variant and the dashed curve of the same color is the projected variant using exactly the same parameters. The shaded region is the 95% confidence interval for the projected curve using a binomial distribution and sample size of* 30*. The mapped curves are from the same evaluations as in Section 5.5. The projected curves are from a separate independent evaluation. Therefore, train/test splits and perturbations are different.*

D.7 CGSPM Runtime: Individual Graph Runtimes

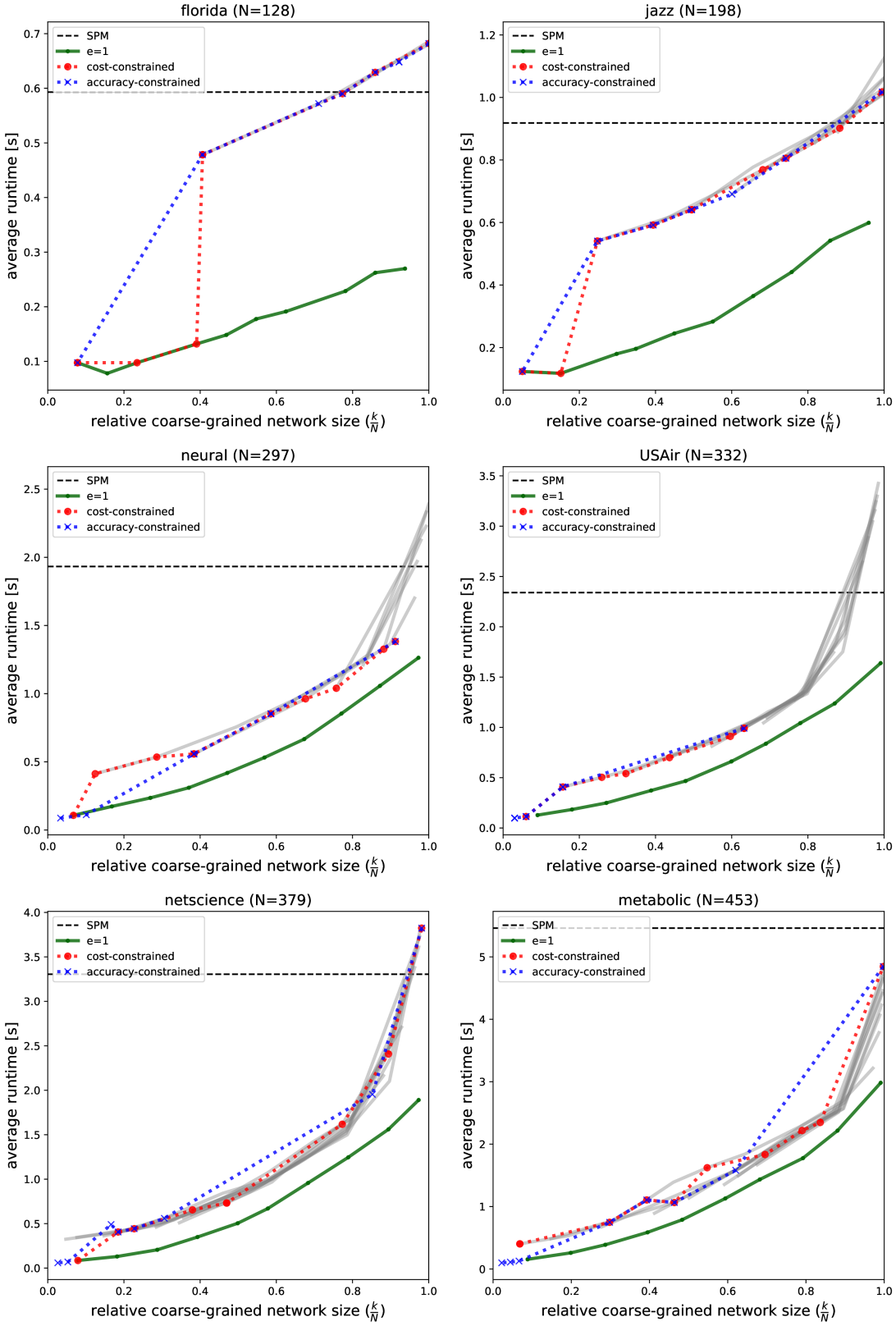


Fig. D.7

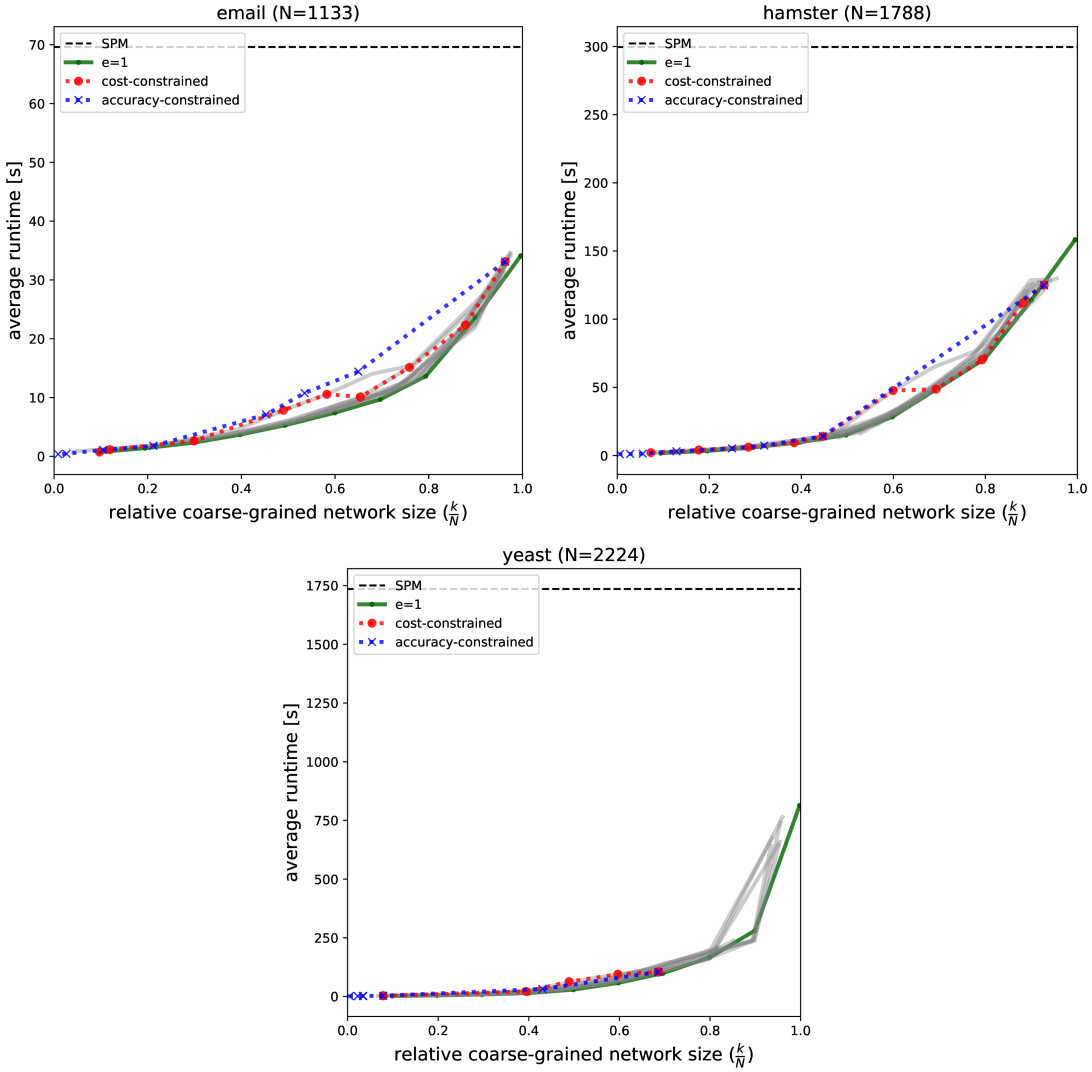


Fig. D.7 (cont.)**:** *Average CGSPM Runtimes as function in k compared to SPM. Dotted curves indicate runtimes with the cost-constrained (red) and accuracy-constrained (blue) optimization strategies. Gray lines show average runtimes sampled from all recorded data. The runtimes with $e = 1$ are highlighted as well.*

D.8 ECSPM ROC Curves

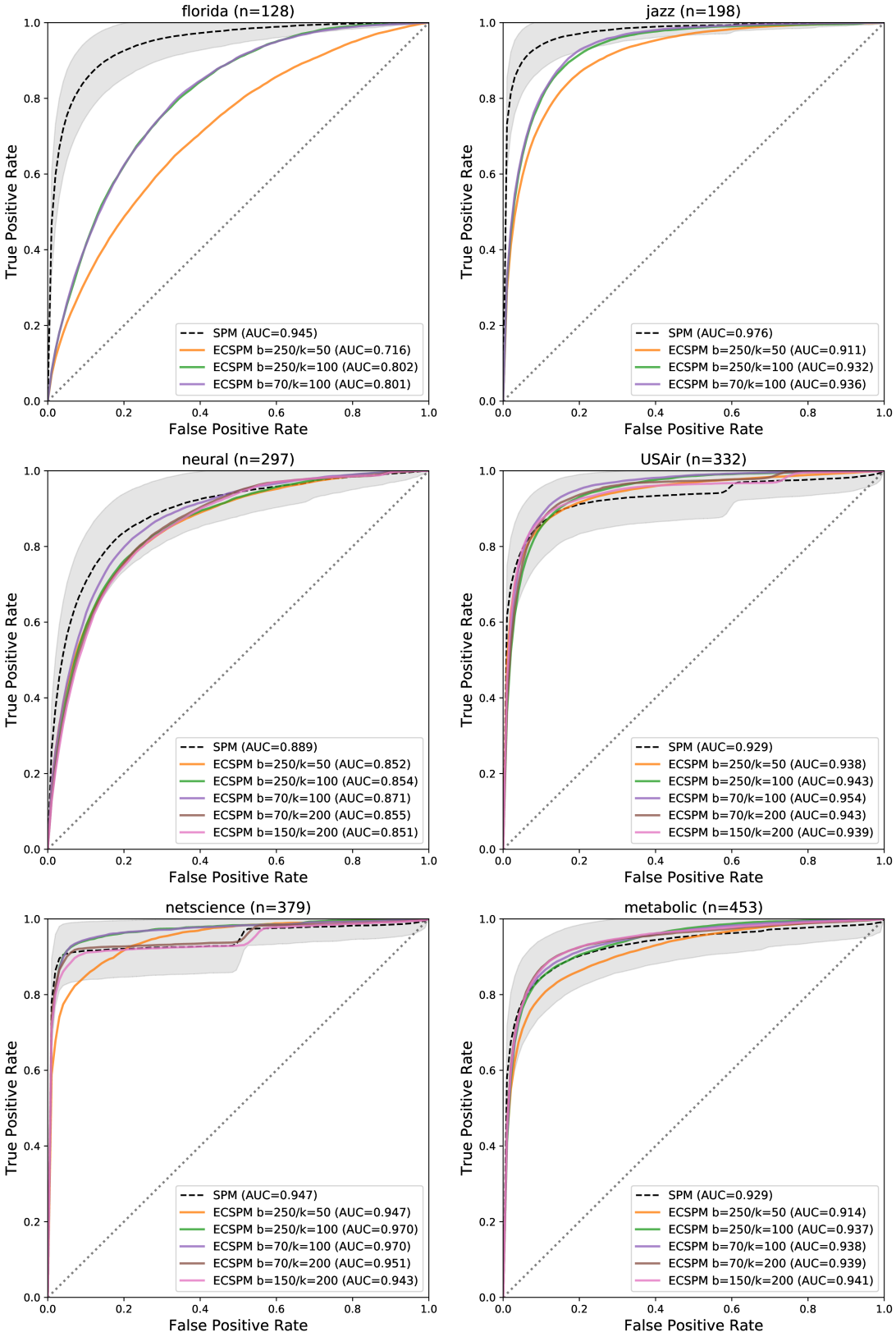


Fig. D.8

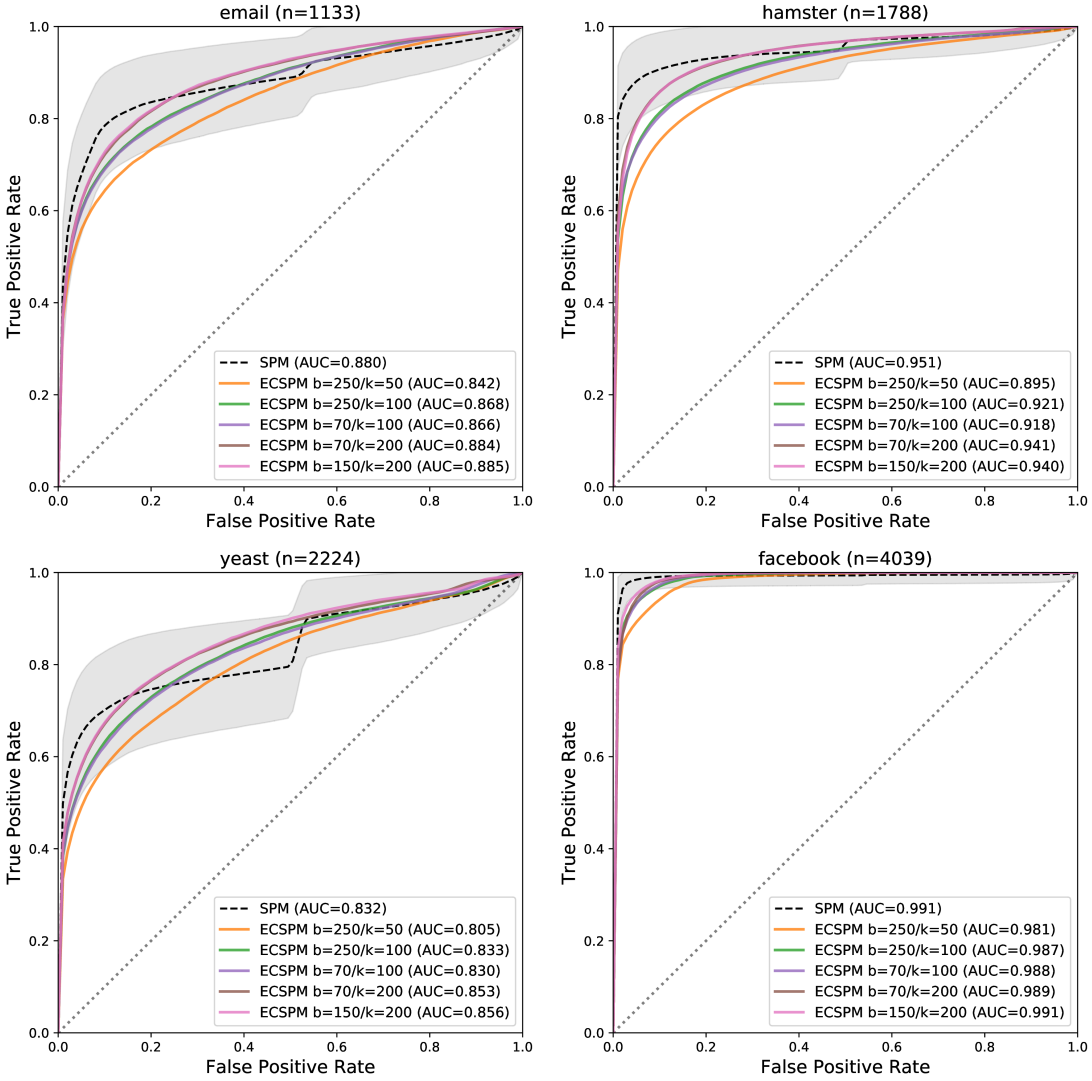

email (n=1133)
True Positive Rate
False Positive Rate
SPM (AUC=0.880)
ECSPM b=250/k=50 (AUC=0.842)
ECSPM b=250/k=100 (AUC=0.868)
ECSPM b=70/k=100 (AUC=0.866)
ECSPM b=70/k=200 (AUC=0.884)
ECSPM b=150/k=200 (AUC=0.885)
hamster (n=1788)
True Positive Rate
False Positive Rate
SPM (AUC=0.951)
ECSPM b=250/k=50 (AUC=0.895)
ECSPM b=250/k=100 (AUC=0.921)
ECSPM b=70/k=100 (AUC=0.918)
ECSPM b=70/k=200 (AUC=0.941)
ECSPM b=150/k=200 (AUC=0.940)
yeast (n=2224)
True Positive Rate
False Positive Rate
SPM (AUC=0.832)
ECSPM b=250/k=50 (AUC=0.805)
ECSPM b=250/k=100 (AUC=0.833)
ECSPM b=70/k=100 (AUC=0.830)
ECSPM b=70/k=200 (AUC=0.853)
ECSPM b=150/k=200 (AUC=0.856)
facebook (n=4039)
True Positive Rate
False Positive Rate
SPM (AUC=0.991)
ECSPM b=250/k=50 (AUC=0.981)
ECSPM b=250/k=100 (AUC=0.987)
ECSPM b=70/k=100 (AUC=0.988)
ECSPM b=70/k=200 (AUC=0.989)
ECSPM b=150/k=200 (AUC=0.991)

Fig. D.8 (cont.)

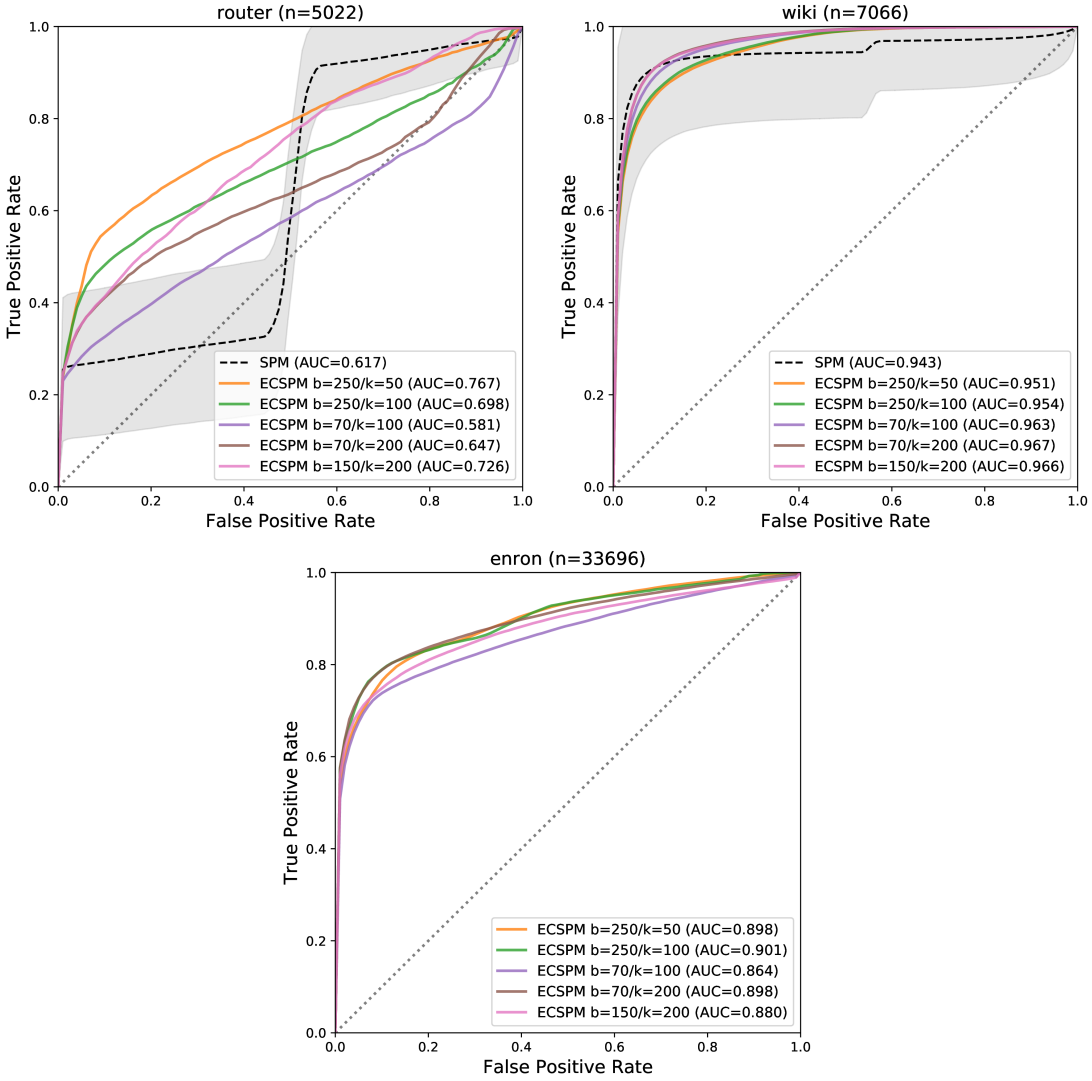


Fig. D.8 (cont.)**:** *ECSPM ROC curves. The gray shaded area around the SPM curve represents the 95% confidence interval obtained by fitting a binomial distribution. Colored curves represent different parameter combinations.*

D.9 ECSPM/CGSPM ROC Curves

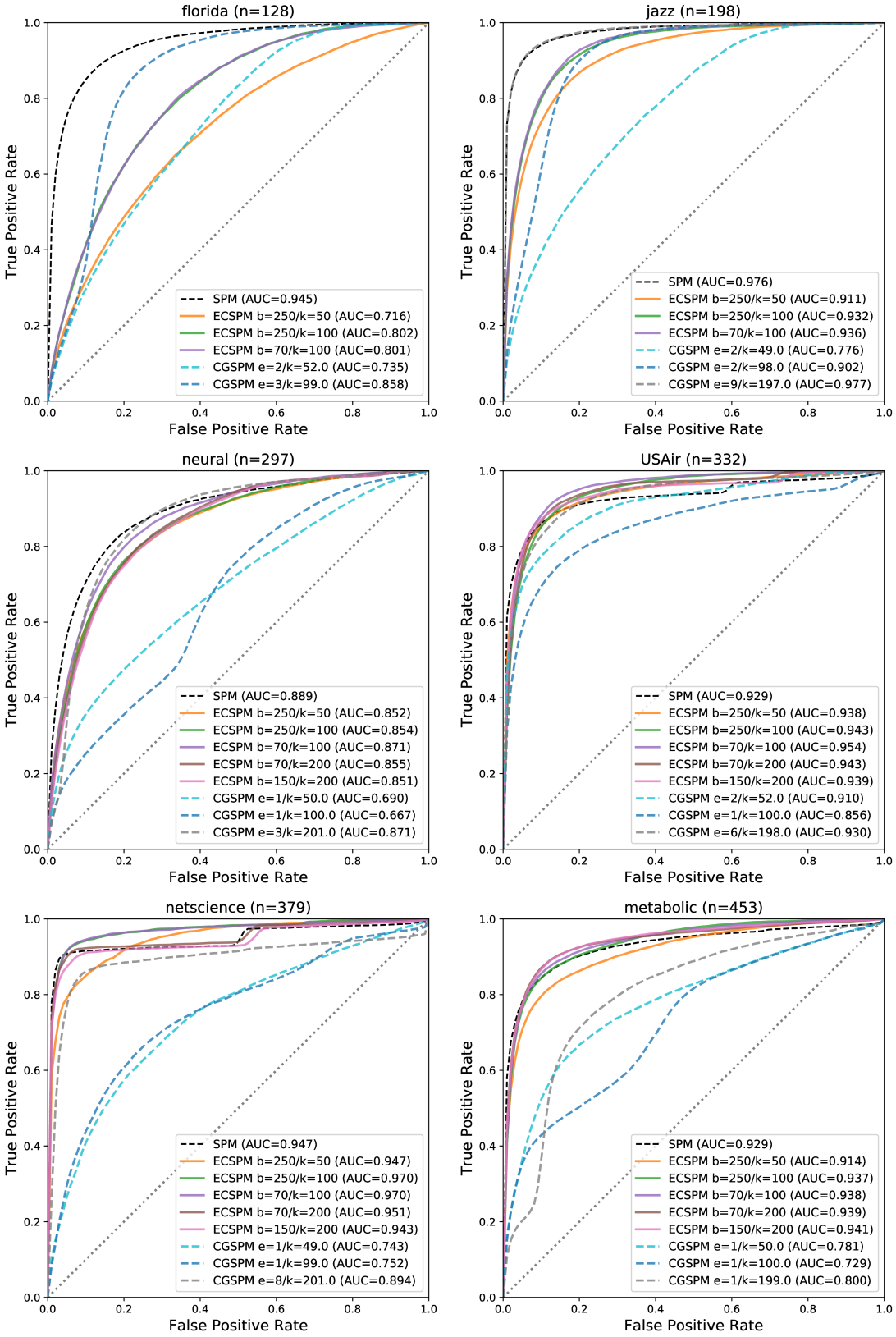


Fig. D.9

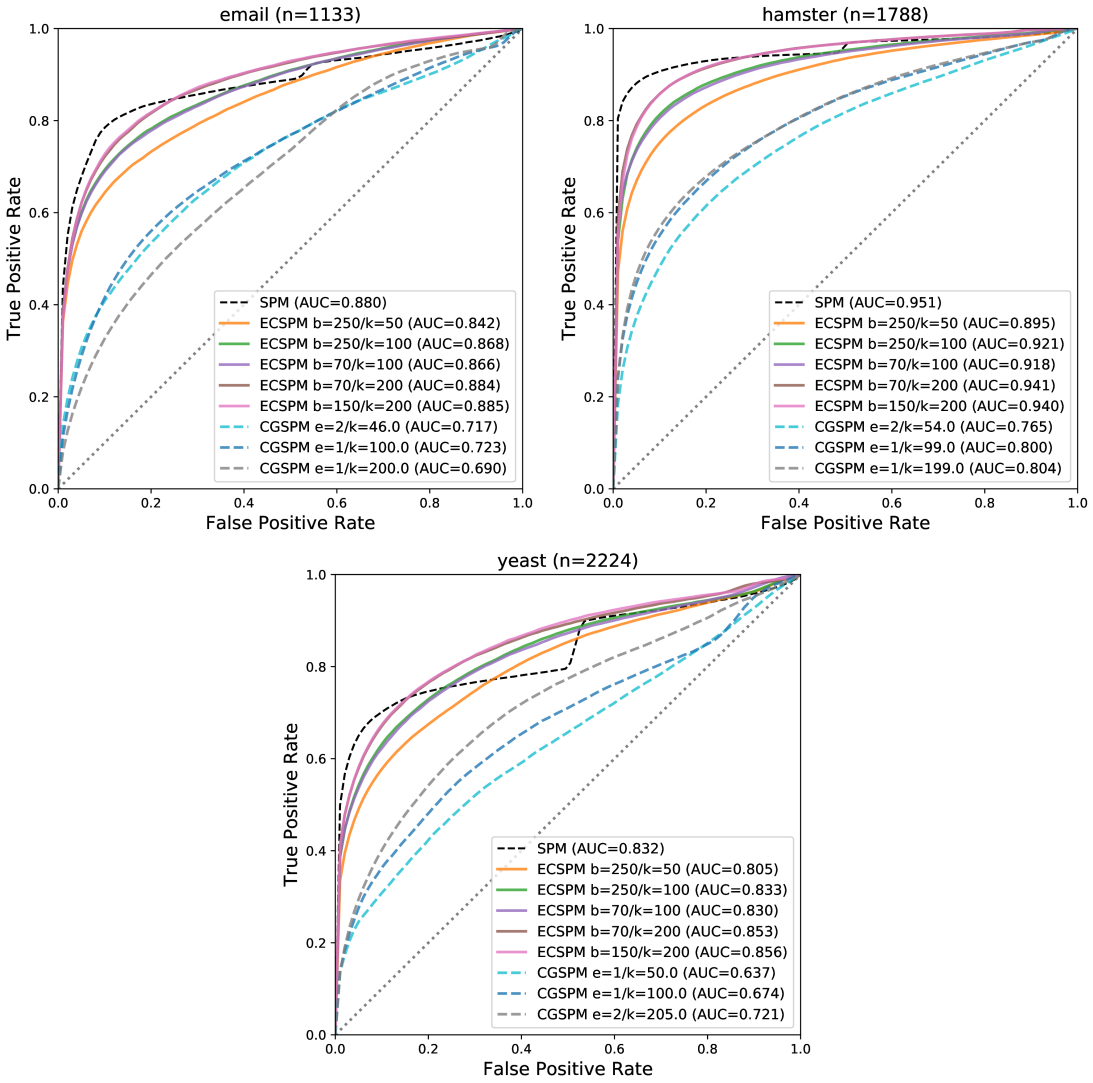


Fig. D.9 (cont.)**:** *ECSPM compared to CGSPM at same coarse-grained graph size.*

D.10 ECSPM Runtime Bar Charts

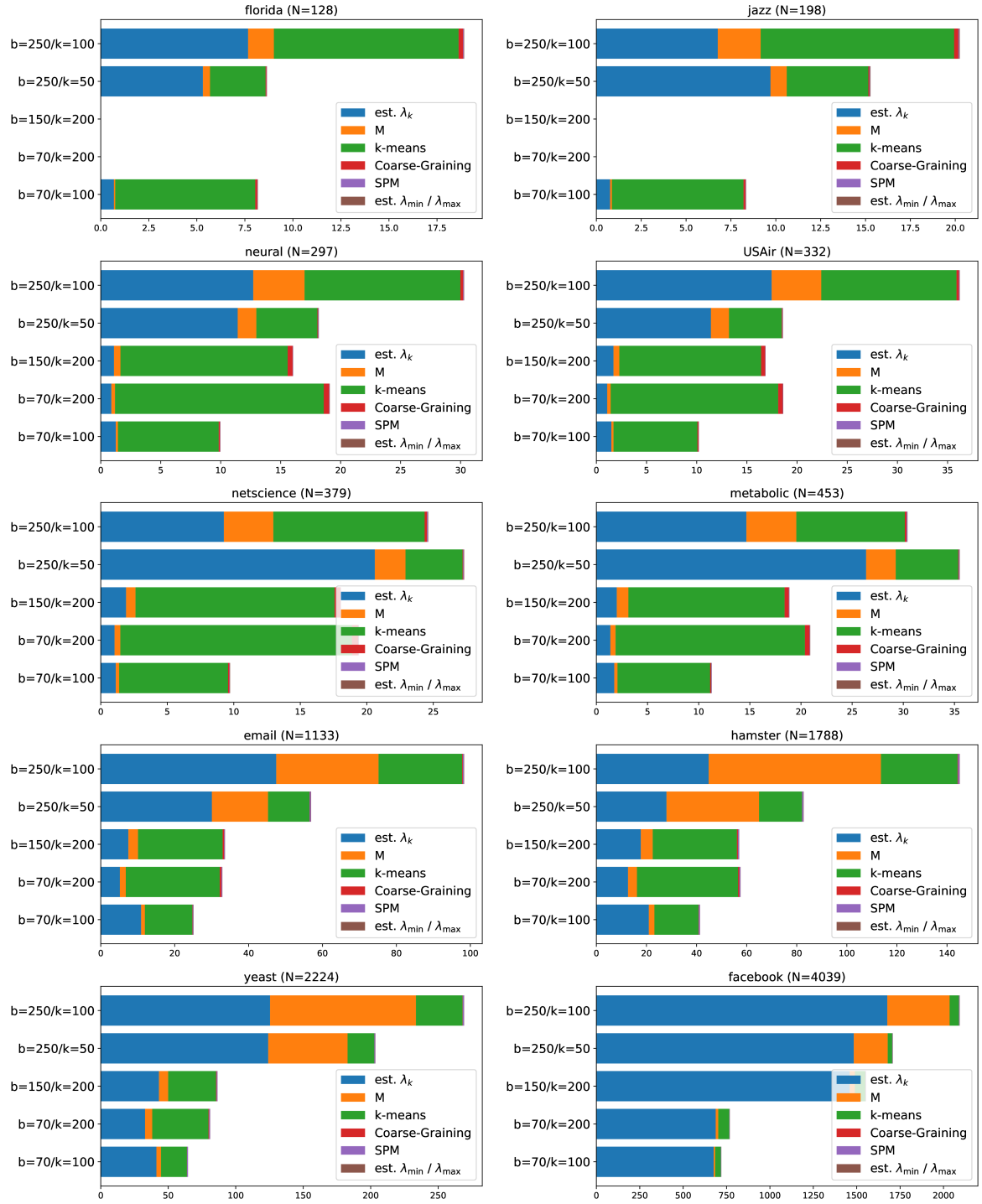


Fig. D.10

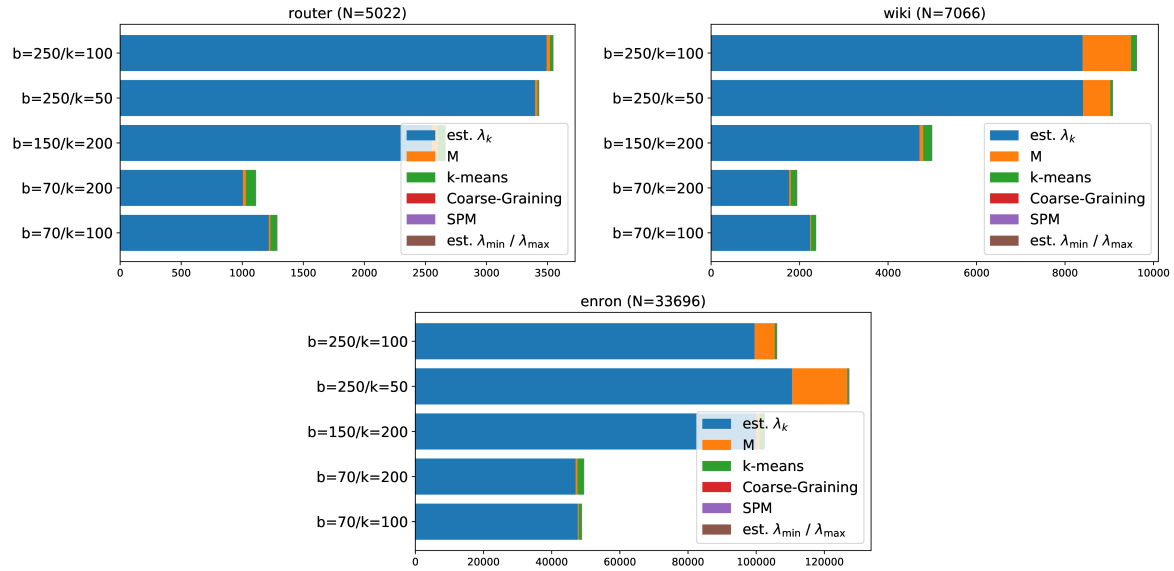


Fig. D.10 (cont.): *Runtime spent of different tasks in ECSPM.*

Appendix E

Supplementary Proofs

E.1 Equal Eigenvectors Components Imply Row Duplication

Row duplication in matrix $\mathbf{A}$ occurs when eigenvector components associated to two given vertices v_l and v_m have the same value in every eigenvector. For the following proof, the association between vertices and eigenvector components as constructed in Section 2.3.4 is used.

Proposition. *Let $\mathbf{A} \in \mathbb{R}^{N\times N}$ be a symmetric adjacency matrix with eigenvectors u_k and $k \in \{1,\dots,N\}$. Given a pair of indices $l, m \in \{1,\dots,N\}$ and $l \neq m$, when $\lambda_k \neq 0$ and $u_k(l) = u_k(m)$ for all k, then the l-th and m-th rows of matrix $\mathbf{A}$ are equal.*

Proof. Any component of matrix $\mathbf{A}$ can be expressed as a linear combination of its eigenpairs (see Section 2.3). The eigendecomposition (2) for a single component of $\mathbf{A}$ reads

$$A_{ij} = \sum_{k=1}^{N} \lambda_k u_k(i) u_k(j)^\top. \tag{90}$$

Suppose the matrix components in two different rows are equal, that is, $A_{lj} = A_{mj}$ and $l \neq m$ for all columns $j \in \{1,\dots,N\}$. Using (90), this relationship can be expressed as

$$\sum_{k=1}^{N} \lambda_k u_k(l) u_k^\top(j) = \sum_{k=1}^{N} \lambda_k u_k(m) u_k^\top(j). \tag{91}$$

Consider only a particular eigenvector $k = n$,

$$\lambda_n u_n(l) u_n^\top(j) = \lambda_n u_n(m) u_n^\top(j). \tag{92}$$

- If $\lambda_n = 0$: The eigenvector u_n does not contribute to the eigendecomposition. This case can be ignored.
- If $\lambda_n \neq 0$: Equality in (92) occurs when components l and m of eigenvector u_1 are equal.

The equality in (92) for all $k \in \{1,\dots,N\}$ eigenvectors is a sufficient condition for the equality in (91) to hold. Furthermore, when the components l and m are equal in each eigenvector with associated non-zero eigenvalue, the rows l and m of $\mathbf{A}$ are necessarily equal as well.

$$\exists l, m\{1,\dots,N\} : l \neq m : [\forall k \in \{1,\dots,N\} : u_k(l) = u_k(m)] \Rightarrow \forall j \in \{1,\dots,N\} : A_{lj} = A_{mj}.$$

□

Note that there reverse implication is not true. Equal matrix components state nothing conclusive about the equality of eigenvector components.

E.2 Maximal Difference Between Real Eigenvector Components

The following proof uses a basic result from spectral graph theory which states that all components of an eigenvector u are real numbers when $\mathbf{A}$ is a real and symmetric matrix. A proof can be found, for example, in Horn and Johnson (1985).

Proposition. *Let u be a unit length eigenvector of a symmetric matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ and let the range of eigenvector components be $\delta_u = \kappa_{\max} - \kappa_{\min}$, where $\kappa_{\max}$ denotes the eigenvector component of u with largest value and $\kappa_{\min}$ denotes the eigevector component of u with smallest value, as defined in Section 4.3.2. Then, $\delta_u \leq \sqrt{2}$.*

Proof. Solve the definition of δ_u for $\kappa_{\max}$:

$$\kappa_{\max} = \delta_u + \kappa_{\min} \tag{93}$$

Suppose $K_r = \{\kappa_1, \ldots, \kappa_N\} \setminus \{\kappa_{\min}, \kappa_{\max}\}$ is the set of all vector components of u without $\kappa_{\min}$ and $\kappa_{\max}$. The eigenvector is normalized to unit length. Therefore, the squared norm is

$$\begin{aligned} \|u\|^2 &= \sum_{i=1}^{N} \kappa_i^2 = 1 \\ \kappa_{\min}^2 + \kappa_{\max}^2 &= 1 - \sum_{\kappa_i \in K_r} \kappa_i^2. \end{aligned} \tag{94}$$

Substitute $\kappa_{\max}$ from (93) and solve for $\kappa_{\min}$ using the quadratic formula:

$$\begin{aligned} \kappa_{\min}^2 + (\delta_u + \kappa_{\min})^2 &= 1 - \sum_{\kappa_i \in K_r} \kappa_i^2 \\ 2\kappa_{\min}^2 + 2\delta_u \kappa_{\min} + \left(\delta_u^2 - 1 + \sum_{\kappa_i \in K_r} \kappa_i^2 \right) &= 0 \\ \kappa_{\min} &= \frac{-2\delta_u \pm \sqrt{-4\delta_u^2 + 8 - 8\sum_{\kappa_i \in K_r} \kappa_i^2}}{4}. \end{aligned} \tag{95}$$

Equation (95) has a unique real solution when the discriminant is at least zero:

$$\begin{aligned} -4\delta_u^2 + 8 - 8 \sum_{\kappa_i \in K_r} \kappa_i^2 &\geq 0 \\ \delta_u &\leq \sqrt{2 - 2 \sum_{\kappa_i \in K_r} \kappa_i^2}. \end{aligned} \tag{96}$$

Note that $\kappa_{\min} \leq \kappa_{\max}$ by definition and due to (94) the term under the root is non-negative. It can be seen immediately that δ_u is maximal when all eigenvector components $\kappa_i \in K_r$ are zero. Then, (96) reduces to $\delta_u \leq \sqrt{2}$. □

List of Figures

List of Tables

List of Algorithms

References

Abdi, H., Williams, L.J.: Principal component analysis. Wiley interdisciplinary reviews: computational statistics 2(4), 433–459 (2010)

Achlioptas, D.: Database-friendly random projections: Johnson-lindenstrauss with binary coins. Journal of computer and System Sciences 66(4), 671–687 (2003)

de Aguiar, M.A.M., Bar-Yam, Y.: Spectral analysis and the dynamic response of complex networks. Phys. Rev. E 71, 016106 (Jan 2005), URL https://link.aps.org/doi/10.1103/PhysRevE.71.016106

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edn. (1999)

Arnoldi, W.E.: The principle of minimized iterations in the solution of the matrix eigenvalue problem. Quarterly of applied mathematics 9(1), 17–29 (1951)

Arthur, D., Vassilvitskii, S.: k-means++: The advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. pp. 1027–1035. Society for Industrial and Applied Mathematics (2007)

Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., van der Vorst, H.: Templates for the solution of algebraic eigenvalue problems: a practical guide. SIAM (2000)

Batagelj, V., Mrvar, A.: Pajek datasets (2006), URL http://vlado.fmf.uni-lj.si/pub/networks/data/

Biggs, N.: Algebraic Graph Theory. Cambridge Mathematical Library, Cambridge University Press, 2 edn. (1974)

Bu, D., Zhao, Y., Cai, L., Xue, H., Zhu, X., Lu, H., Zhang, J., Sun, S., Ling, L., Zhang, N., et al.: Topological structure analysis of the protein–protein interaction network in budding yeast. Nucleic acids research 31(9), 2443–2450 (2003)

Calvetti, D., Reichel, L., Sorensen, D.C.: An implicitly restarted lanczos method for large symmetric eigenvalue problems. Electronic Transactions on Numerical Analysis 2(1), 21 (1994)

Carslaw, H.S.: Introduction to the Theory of Fourier's Series and Integrals. Macmillan (1921)

Champagne, B.: Adaptive eigendecomposition of data covariance matrices based on first-order perturbations. IEEE Transactions on Signal Processing 42(10), 2758–2770 (1994)

Cheeger, J.: A lower bound for the smallest eigenvalue of the laplacian. In: Proceedings of the Princeton conference in honor of Professor S. Bochner (1969)

Chen, J., Lu, J.a., Lu, X., Wu, X., Chen, G.: Spectral coarse graining of complex clustered networks. Communications in Nonlinear Science and Numerical Simulation 18(11), 3036–3045 (2013)

Cvetković, D., Čangalović, M., Kovačević-Vujčić, V.: Semidefinite programming methods for the symmetric traveling salesman problem. In: International Conference on Integer Programming and Combinatorial Optimization. pp. 126–136. Springer (1999)

Cvetković, D.M., Doob, M., Sachs, H.: Spectra of graphs: theory and application, vol. 87. Academic Pr (1980)

Cvetkovic, D., Rowlinson, P., Simic, S.: An introduction to the theory of graph spectra. London Mathematical Society Student Texts, Cambridge University Press, Cambridge (2009), URL http://cds.cern.ch/record/1614369

Cvetković, D., Simić, S.: Graph spectra in Computer Science. Linear Algebra and its Applications 434(6), 1545–1562 (Mar 2011), URL https://linkinghub.elsevier.com/retrieve/pii/S0024379510006117

Dasgupta, S., Gupta, A.: An elementary proof of the Johnson-Lindenstrauss Lemma. Tech. rep. (1999)

Davis, C., Kahan, W.: The rotation of eigenvectors by a perturbation. iii. SIAM Journal on Numerical Analysis 7(1), 1–46 (1970), URL https://doi.org/10.1137/0707001

Demmel, J.W.: Applied numerical linear algebra, vol. 56. Siam (1997)

Demmel, J.W., Marques, O.A., Parlett, B.N., Vömel, C.: Performance and accuracy of lapack's symmetric tridiagonal eigensolvers. SIAM Journal on Scientific Computing 30(3), 1508–1526 (2008)

Di Napoli, E., Polizzi, E., Saad, Y.: Efficient estimation of eigenvalue counts in an interval. Numerical Linear Algebra with Applications 23(4), 674–692 (2016)

Duch, J., Arenas, A.: Community detection in complex networks using extremal optimization. Physical review E 72(2), 027104 (2005)

Dunnett, C.W.: A multiple comparison procedure for comparing several treatments with a control. Journal of the American Statistical Association 50(272), 1096–1121 (1955)

Egan, J.P.: Signal Detection Theory and ROC Analysis Academic Press Series in Cognition and Perception. London, UK: Academic Press (1975)

Eldridge, J., Belkin, M., Wang, Y.: Unperturbed: spectral analysis beyond davis-kahan. arXiv preprint arXiv:1706.06516 (2017)

Faber, G.: Über tschebyscheffsche polynome. Journal für die reine und angewandte Mathematik 150, 79–106 (1920)

Farkas, I.J., Derényi, I., Barabási, A.L., Vicsek, T.: Spectra of "real-world" graphs: Beyond the semicircle law. Phys. Rev. E 64, 026704 (Jul 2001), URL https://link.aps.org/doi/10.1103/PhysRevE.64.026704

Fawcett, T.: Roc graphs: Notes and practical considerations for researchers. Machine learning 31(1), 1–38 (2004)

Gfeller, D.: Simplifying complex networks: from a clustering to a coarse graining strategy. Phd dissertation, EPFL Lausanne (2007), URL http://infoscience.epfl.ch/record/109759

Gfeller, D., De Los Rios, P.: Spectral coarse graining of complex networks. Phys. Rev. Lett. 99, 038701 (07 2007), URL https://link.aps.org/doi/10.1103/PhysRevLett.99.038701

Gfeller, D., De Los Rios, P.: Spectral coarse graining and synchronization in oscillator networks. Physical review letters 100(17), 174104 (2008)

Gleiser, P.M., Danon, L.: Community structure in jazz. Advances in Complex Systems 6(4), 565–573 (2003)

Greenbaum, A., Trefethen, L.N.: GMRES/CR and Arnoldi/Lanczos as Matrix Approximation Problems. SIAM Journal on Scientific Computing 15(2), 359–368 (Mar 1994), URL http://epubs.siam.org/doi/10.1137/0915025

Guimerà, R., Danon, L., Díaz-Guilera, A., Giralt, F., Arenas, A.: Self-similar community structure in a network of human interactions. Phys. Rev. E 68(6), 065103 (2003)

Guimerà, R., Sales-Pardo, M.: Missing and spurious interactions and the reconstruction of complex networks. Proceedings of the National Academy of Sciences 106(52), 22073–22078 (2009)

Hammond, D.K., Vandergheynst, P., Gribonval, R.: Wavelets on graphs via spectral graph theory. Applied and Computational Harmonic Analysis 30(2), 129–150 (Mar 2011), URL http://www.sciencedirect.com/science/article/pii/S1063520310000552

Hanley, J.A., McNeil, B.J.: A method of comparing the areas under receiver operating characteristic curves derived from the same cases. Radiology 148(3), 839–843 (1983)

Hartigan, J.A., Wong, M.A.: Algorithm as 136: A k-means clustering algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics) 28(1), 100–108 (1979)

Higham, N.: Functions of Matrices. Other Titles in Applied Mathematics, Society for Industrial and Applied Mathematics (Jan 2008), URL https://epubs.siam.org/doi/book/10.1137/1.9780898717778

Horn, R.A., Johnson, C.R.: Matrix Analysis. Cambridge University Press (1985)

Hutchinson, M.: A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. Communications in Statistics - Simulation and Computation 19(2), 433–450 (1990), URL https://doi.org/10.1080/03610919008812866

Indyk, P., Motwani, R.: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing. pp. 604–613. STOC '98, ACM, New York, NY, USA (1998), URL http://doi.acm.org/10.1145/276698.276876, event-place: Dallas, Texas, USA

Jiao, P., Cai, F., Feng, Y., Wang, W.: Link predication based on matrix factorization by fusion of multi class organizations of the network. Scientific Reports 7(1), 8937 (2017)

Johnson, W.B., Lindenstrauss, J.: Extensions of Lipschitz mappings into a Hilbert space. In: Beals, R., Beck, A., Bellow, A., Hajian, A. (eds.) Contemporary Mathematics, vol. 26, pp. 189–206. American Mathematical Society, Providence, Rhode Island (1984), URL http://www.ams.org/conm/026/

Katz, L.: A new status index derived from sociometric analysis. Psychometrika 18(1), 39–43 (1953)

Klimt, B., Yang, Y.: Introducing the enron corpus. In: CEAS (2004)

Kunegis, J.: Konect: The koblenz network collection. In: Proceedings of the 22nd International Conference on World Wide Web. pp. 1343–1350. ACM (2013)

de Lachapelle, D.M., Gfeller, D., De Los Rios, P.: Shrinking matrices while preserving their eigenpairs with application to the spectral coarse graining of graphs. Submitted to SIAM Journal on Matrix Analysis and Applications (2008)

Lanczos, C.: An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. United States Governm. Press Office Los Angeles, CA (1950)

Lenth, R.V.: Least-squares means: The R package lsmeans. Journal of Statistical Software 69(1), 1–33 (2016)

Leskovec, J., Faloutsos, C.: Sampling from large graphs. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 631–636. ACM (2006)

Leskovec, J., Huttenlocher, D., Kleinberg, J.: Predicting positive and negative links in online social networks. In: Proceedings of the 19th international conference on World wide web. pp. 641–650. ACM (2010a)

Leskovec, J., Huttenlocher, D., Kleinberg, J.: Signed networks in social media. In: Proceedings of the SIGCHI conference on human factors in computing systems. pp. 1361–1370. ACM (2010b)

Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (jun 2014)

Leskovec, J., Lang, K.J., Dasgupta, A., Mahoney, M.W.: Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. Internet Mathematics 6(1), 29–123 (2009)

Leskovec, J., Mcauley, J.J.: Learning to discover social circles in ego networks. In: Advances in neural information processing systems. pp. 539–547 (2012)

Liben-Nowell, D., Kleinberg, J.: The link-prediction problem for social networks. Journal of the American society for information science and technology 58(7), 1019–1031 (2007)

Lloyd, S.: Least squares quantization in pcm. IEEE Transactions on Information Theory 28(2), 129–137 (March 1982)

Lü, L., Pan, L., Zhou, T., Zhang, Y.C., Stanley, H.E.: Toward link predictability of complex networks. Proceedings of the National Academy of Sciences 112(8), 2325–2330 (2015), URL http://www.pnas.org/content/112/8/2325.abstract

Lü, L., Zhou, T.: Link prediction in complex networks: A survey. Physica A: Statistical Mechanics and its Applications 390(6), 1150–1170 (2011)

MacArthur, B.D., Sánchez-García, R.J.: Spectral characteristics of network redundancy. Physical Review E 80(2), 026117 (2009)

Macskassy, S., Provost, F.: Confidence bands for roc curves: Methods and an empirical study. Proceedings of the First Workshop on ROC Analysis in AI. August 2004. (2004)

Mason, J.C., Handscomb, D.C.: Chebyshev polynomials. Chapman and Hall/CRC (2002)

Muscoloni, A., Cannistraci, C.V.: Local-ring network automata and the impact of hyperbolic geometry in complex network link-prediction. arXiv preprint arXiv:1707.09496 (2017)

Newman, M.E.: Finding community structure in networks using the eigenvectors of matrices. Physical review E 74(3), 036104 (2006)

Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab (1999)

Pan, L., Zhou, T., Lü, L., Hu, C.K.: Predicting missing links and identifying spurious links via likelihood analysis. Scientific reports 6, 22955 (2016)

Pan, V.Y., Chen, Z., Zheng, A., et al.: The complexity of the algebraic eigenproblem. Mathematical Sciences Research Institute, Berkeley pp. 1998–71 (1998)

Paratte, J., Martin, L.: Fast eigenspace approximation using random signals. arXiv preprint arXiv:1611.00938 (2016)

Pech, R., Hao, D., Pan, L., Cheng, H., Zhou, T.: Link prediction via matrix completion. EPL (Europhysics Letters) 117(3), 38002 (feb 2017), URL https://doi.org/10.1209%2F0295-5075%2F117%2F38002

Perraudin, N., Paratte, J., Shuman, D., Martin, L., Kalofolias, V., Vandergheynst, P., Hammond, D.K.: GSPBOX: A toolbox for signal processing on graphs. ArXiv e-prints (Aug 2014)

Preciado, V.M., Rahimian, M.A.: Moment-based spectral analysis of random graphs with given expected degrees. IEEE Transactions on Network Science and Engineering 4(4), 215–228 (Oct 2017)

Provost, F., Fawcett, T.: Robust classification systems for imprecise environments. In: AAAI/IAAI. pp. 706–713 (1998)

Raghupathi, K.: 10 interesting use cases for the k-means algorithm (Mar 2018), URL https://dzone.com/articles/10-interesting-use-cases-for-the-k-means-algorithm

Ramasamy, D., Madhow, U.: Compressive spectral embedding: sidestepping the SVD. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 550–558. Curran Associates, Inc. (2015), URL http://papers.nips.cc/paper/5992-compressive-spectral-embedding-sidestepping-the-svd.pdf

Rellich, F., Berkowitz, J.: Perturbation theory of eigenvalue problems. CRC Press (1969)

Saad, Y.: Numerical methods for large eigenvalue problems: revised edition, vol. 66. Siam (2011)

Sandryhaila, A., Moura, J.M.: Discrete signal processing on graphs. IEEE transactions on signal processing 61(7), 1644–1656 (2013)

Schofield, G., Chelikowsky, J.R., Saad, Y.: A spectrum slicing method for the Kohn–Sham problem. Computer Physics Communications 183(3), 497–505 (Mar 2012), URL http://www.sciencedirect.com/science/article/pii/S0010465511003675

Shuman, D.I., Narang, S.K., Frossard, P., Ortega, A., Vandergheynst, P.: The Emerging Field of Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Domains. IEEE Signal Processing Magazine 30(3), 83–98 (May 2013), URL http://arxiv.org/abs/1211.0053, arXiv: 1211.0053

Shuman, D.I., Vandergheynst, P., Frossard, P.: Chebyshev polynomial approximation for distributed signal processing. In: Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on. pp. 1–8. IEEE (2011)

Silver, R., Roeder, H., Voter, A., Kress, J.: Kernel Polynomial Approximations for Densities of States and Spectral Functions. Journal of Computational Physics 124(1), 115–130 (Mar 1996), URL http://linkinghub.elsevier.com/retrieve/pii/S0021999196900480

Sorensen, D.C.: Implicit application of polynomial filters in a k-step arnoldi method. Siam journal on matrix analysis and applications 13(1), 357–385 (1992)

Toh, K.C., Trefethen, L.N.: The Chebyshev Polynomials of a Matrix. SIAM Journal on Matrix Analysis and Applications 20(2), 400–419 (Jan 1998), URL http://epubs.siam.org/doi/10.1137/S0895479896303739

Trefethen, L.N.: Approximation theory and approximation practice, vol. 128. Siam (2013)

Trefethen, L.N., Bau III, D.: Numerical linear algebra, vol. 50. Siam (1997)

Tremblay, N., Puy, G., Borgnat, P., Gribonval, R., Vandergheynst, P.: Accelerated spectral clustering using graph filtering of random signals. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 4094–4098. Ieee (2016a)

Tremblay, N., Puy, G., Gribonval, R., Vandergheynst, P.: Compressive spectral clustering. In: International Conference on Machine Learning. pp. 1002–1011 (2016b)

Ulanowicz, R., Bondavalli, C., Egnotovich, M.: Network analysis of trophic dynamics in south florida ecosystem, fy 97: The florida bay ecosystem. Annual Report to the United States Geological Service Biological Resources Division Ref. No.[UMCES] CBL pp. 98–123 (1998)

Wang, T., Liao, G.: A review of link prediction in social networks. In: Management of e-Commerce and e-Government (ICMeCG), 2014 International Conference on. pp. 147–150. IEEE (2014)

Wang, T., He, X.S., Zhou, M.Y., Fu, Z.Q.: Link prediction in evolving networks based on popularity of nodes. Scientific reports 7(1), 7147 (2017)

Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world'networks. nature 393(6684), 440–442 (1998)

Weierstrass, K.: Über die analytische darstellbarkeit sogenannter willkürlicher functionen einer reellen veränderlichen. Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin 2, 633–639 (1885)

Weyl, H.: Das asymptotische verteilungsgesetz der eigenwerte linearer partieller differentialgleichungen (mit einer anwendung auf die theorie der hohlraumstrahlung). Mathematische Annalen 71(4), 441–479 (1912)

Wilbraham, H.: On a certain periodic function. The Cambridge and Dublin Mathematical Journal 3, 198–201 (1848)

Xu, X., Liu, B., Wu, J., Jiao, L.: Link prediction in complex networks via matrix perturbation and decomposition. Scientific reports 7(1), 14724 (2017)

Ying, X., Wu, X.: Randomizing social networks: a spectrum preserving approach. In: proceedings of the 2008 SIAM International Conference on Data Mining. pp. 739–750. SIAM (2008)

Zeng, L., Jia, Z., Wang, Y.: Influence of topological properties of complex networks on the effect of spectral coarse-grained network. Communications and Network 10(03), 93 (2018a)

Zeng, L., Jia, Z., Wang, Y.: A new spectral coarse-graining algorithm based on k-means clustering in complex networks. Modern Physics Letters B p. 1850421 (2018b)

Zeng, X., Liu, L., Lü, L., Zou, Q., Valencia, A.: Prediction of potential disease-associated micrornas using structural perturbation method. Bioinformatics 1, 8 (2018c)

Curriculum Vitae

Personal Data

Born	19.12.1983 in Zürich, Switzerland
Address	Schwandenwiesen 26, 8052 Zurich

Education

09/2013 – 09/2019 **Doctoral program at the University of Zurich**, *Department of Informatics*.
Chair of Dynamic and Distributed Information Systems Group

08/2011 – 08/2013 **Master of Science in Informatics**, *University of Zurich*.
Specialization: Software Systems.

11/2006 – 07/2011 **Bachelor of Science in Informatics**, *University of Zurich*.
Specialization: Information Systems.

Professional Experience

08/2019 – present **Data Scientist/Data Engineer**, *ELCA Informatik AG*, Zurich.

09/2013 – 07/2019 **Research Assistant**, *University of Zurich*, Distributed and Dynamic Systems Group.

10/2012 – 08/2013 **Java/PHP Application Developer**, *Getunik AG*, Zurich.

11/2010 – 09/2012 **Java Application Developer**, *Trialox AG*, Zurich.